5-2012

Mitigating Insider Threat in Relational Database Systems

Qussai Yaseen
University of Arkansas, Fayetteville

MITIGATING INSIDER THREAT IN RELATIONAL DATABASE SYSTEMS

MITIGATING INSIDER THREAT IN RELATIONAL DATABASE SYSTEMS

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

By

Qussai Yaseen
Yarmouk University
Bachelor of Science in Computer Science, 2002
Jordan University of Science and Technology
Master of Science in Computer Science, 2006

May 2012
University of Arkansas

ABSTRACT

The dissertation concentrates on addressing the factors and capabilities that enable insiders to violate systems security. It focuses on modeling the accumulative knowledge that insiders get throughout legal accesses, and it concentrates on analyzing the dependencies and constraints among data items and represents them using graph-based methods. The dissertation proposes new types of Knowledge Graphs (KGs) to represent insiders' knowledgebases. Furthermore, it introduces the Neural Dependency and Inference Graph (NDIG) and Constraints and Dependencies Graph (CDG) to demonstrate the dependencies and constraints among data items. The dissertation discusses in detail how insiders use knowledgebases and dependencies and constraints to get unauthorized knowledge. It suggests new approaches to predict and prevent the aforementioned threat. The proposed models use KGs, NDIG and CDG in analyzing the threat status, and leverage the effect of updates on the lifetimes of data items in insiders' knowledgebases to prevent the threat without affecting the availability of data items. Furthermore, the dissertation uses the aforementioned idea in ordering the operations of concurrent tasks such that write operations that update risky data items in knowledgebases are executed before the risky data items can be used in unauthorized inferences. In addition to unauthorized knowledge, the dissertation discusses how insiders can make unauthorized modifications in sensitive data items. It introduces new approaches to build Modification Graphs that demonstrate the authorized and unauthorized data items which insiders are able to update. To prevent this threat, the dissertation provides two methods, which are hiding sensitive dependencies and denying risky write requests. In addition to traditional RDBMS, the dissertation investigates insider threat in cloud relational database systems (cloud RDMS). It discusses the vulnerabilities in the cloud computing structure that may enable insiders to launch

attacks. To prevent such threats, the dissertation suggests three models and addresses the advantages and limitations of each one.

To prove the correctness and the effectiveness of the proposed approaches, the dissertation uses well stated algorithms, theorems, proofs and simulations. The simulations have been executed according to various parameters that represent the different conditions and environments of executing tasks.

This dissertation is approved for recommendation
to the Graduate Council

Dissertation Director:

_____

Dr. Brajendra Panda

Dissertation Committee:

_____

Dr. Gordon Beavers

_____

Dr. Dale R. Thompson

_____

Dr. David Douglas

DISSERTATION DUPLICATION RELEASE

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

Agreed _____
Qussai Yaseen

Refused _____
Qussai Yaseen

ACKNOLEDGMENTS

I thank my advisor Professor Brajendra Panda for his support of my research, and for his great guidance and help during pursuing my PhD program. I also would like to thank Professor Gordon Beavers, Professor Dale Thompson and Professor David Douglas for serving on my committee; it is a complete privilege having you on my dissertation committee. I also would like to thank my brother Baderaddin Yassin for his great support before and during pursuing my PhD program.

DEDICATION

To my late father… to his pure spirit and heart.

To my mother … to her big heart.

To my brothers and sisters … to this great family.

It is you … the motivation who made this possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Protecting information is as important as protecting other organizational sensitive assets such as money. In this era of rapid revolution in computer technologies and communications, attacks on this vital resource are getting more complicated and harmful. Thus, information security has become a crucial goal to organizations and individuals. Information security means protecting information and information systems from interception, interruption, modification, and fabrication in order to preserve the confidentiality, availability and integrity of information [LII][NIST95]. Confidentiality means hiding information or resources from unauthorized users in order to protect the personnel information privacy, whereas availability means providing timely access to information and information resources. Availability is an important part of systems' reliability and design since limited availability of a system is as bad as if the system does not exist. Information integrity indicates protecting data against inappropriate or unauthorized modifications. Moreover, it means ensuring the accuracy of sources of information, which is called information authenticity [Bishop03] [LII].

Research in information security is generally focused on two fields, network security and databases security. Network security deals with mechanisms that protect information during transmission via networks, while database security represents the methods that protect stored information in DBMS. The dissertation concentrates on database security and proposes methods to protect relational database systems from insider threat.

1.1 Database Security

Database security is a major field in computer security. Databases are targeted by tremendous types of attacks that aim to breach the security of the high value assets that databases store. Threats to databases could be physical, such as theft and destroying physical storage, or could be logical, which are categorized as follows [Baraani96].

a) Unauthorized modifications of information attacks: This type of threat can be launched accidentally by authorized users or intentionally by legal or illegal users. Notice that this type affects information integrity.
b) Exposure of information attacks: Information leaks can occur by direct access or indirect access to information. Indirect exposure of information is performed by inferring the values of unauthorized data items using authorized data items. Hence, information disclosure affects the confidentiality of information.
c) Denial of service attacks: These attacks can be launched by controlling or dominating resources such that other users in a system cannot access them. For example, attackers may consume the computational resources, such as bandwidth, disk space, or processor time, preventing any work from being done.

Significant research has been performed to secure databases against attacks by identifying proper security policies and mechanisms. A security policy represents what is expected from a security system, whereas a security mechanism demonstrates how to achieve security goals. Security policies should have some features or properties that should be satisfied by security mechanisms. The properties that a security policy in databases should have include *access control, inference, consistency, accountability,* and *user identification*. Access control guarantees that direct

accesses to objects are granted according to predefined privileges. In addition, a security policy should protect sensitive information from exposure by unauthorized users using indirect access or inference. Consistency preserves the integrity of databases, and accountability represents the requirements to record all accesses to database objects by users. Accountability is an important feature for preserving the consistency of databases. Finally, user authentication consists of system requirements that lead to the correct identification of legal users in the system [Baraani96][Bishop03].

Extensive research has been done to address mechanisms that achieve the goals of security policies. The access matrix model is an example of access control mechanisms. It was introduced by Lampson [Lampson71] and improved later by other researchers [Conway72] [Harrison76] to manage accesses to resources. The access matrix model uses three components to organize accesses, which are subjects, objects and privileges. Subjects indicate users, which are represented by the rows of the matrix, objects indicate resources and are represented by the columns of the matrix, and privileges indicate permissions of read, write, execute … etc. Privileges are represented inside the cells of the matrix. Figure 1.1 shows an example of an access matrix model, where $Pr(Sub_i, Obj_j)$ represents the privileges that Sub_i have on the object Obj_j.

		Objects				
		Obj_1	Obj_2	Obj_3	….	Obj_n
Subjects	Sub_1	$Pr(Sub_1, Obj_1)$	$Pr(Sub_1, Obj_2)$	$Pr(Sub_1, Obj_3)$	….	$Pr(Sub_1, Obj_n)$
	Sub_2	$Pr(Sub_2, Obj_1)$	$Pr(Sub_2, Obj_2)$	$Pr(Sub_2, Obj_3)$	….	$Pr(Sub_2, Obj_n)$
	.	.	.	.	…	
	Sub_m	$Pr(Sub_m, Obj_1)$	$Pr(Sub_m, Obj_2)$	$Pr(Sub_m, Obj_3)$	….	$Pr(Sub_m, Obj_n)$

Figure 1.1. An Access Matrix

Inference control has been discussed broadly, and many models have been addressed to prevent unauthorized inference of information. Inference channels can be eliminated during database design by building good and secure schema, or they can be removed by evaluating queries to ensure that they do not lead to illegal inference. Biskup et. al [Biskup08] proposed mechanisms and constraints that reduce inference problem in relational databases to access control, in which inference can be controlled. Woodruff and Staddon [David04] introduced the private inference control (PIC), which provides inference control on the server side without learning the information that is retrieved. Hence, inference problem is discussed in detail in the related work chapter. Integrity and authentication can be assured using different techniques such as cryptographic techniques. For example, RSA [Rivest78] is used as a public key authentication. Similarly, encryption methods such as Tripple-DES [DES] and AES [AES] are used to ensure information integrity.

Databases have become increasingly vulnerable to attack due to the vast and continuous revolution in communications and new technologies such as cloud computing. Cloud computing refers to the use of the internet to host computer resources instead of keeping them on local computers. It delivers services (applications) over the internet and the hardware and systems in data centers [Armbrust09]. Applications are hosted and accessible through datacenters. Resources on the cloud are sold (leased) on demand. The price of leasing resources depends on the time duration and the amount and the type of resources needed. Users can have what they want from resources at any time. Resources in the cloud are fully managed by the cloud resources providers. The management of resources includes monitoring, provisioning, de-provisioning, workload balancing, and changing requests [Boss]. The services provided by the

cloud include Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). Moreover, the Amazon cloud enables insiders to create and manage a relational database, which is called Amazon Relational Database Services (RDS) [RDS]. Cloud databases induce new security challenges since sensitive data are migrated to the servers of the cloud providers. Moreover, databases are managed and maintained by the employees of services providers. This maximizes threat, especially insider threat, which is discussed in the next section.

1.2 Insider Threat

Insider threat is a critical security problem. The threat of insiders can be posed intentionally by malicious insiders or unintentionally. Malicious insider threat is defined as the threat that is caused by a person who has authorized access privileges and knowledge of the computer systems of an organization, and is inspired to antagonistically influence the organization [Brackney04]. For the rest of the dissertation, we will use the term "insider threat" to indicate malicious insider threat. We define insider threat according to relational database systems, which is the context of the dissertation, as follows.

Definition 1 (Insider Threat). *Insider threat is the threat that is posed by a person who has authorized access and knowledge of the relational database system s/he uses, is familiar with the dependencies and constraints among data items, and is motivated to violate the security policy of the system throughout authorized access.*

Insiders could be employees, contractors, or business partners. They have the capabilities, which outsiders do not have, that enable them to launch attacks. In the context of relational database

systems, insiders are familiar with the structure of the relational database systems on which they are working. That is, they are familiar with the dependencies and constraints among data items, the sensitive information and the insensitive information, and the inference channels insiders can exploit to infer unauthorized information. Insiders get knowledge during their work on organizations' systems. They can get a part of the knowledge through their activities and transactions in systems, and they get other parts by collaborating with other insiders in the systems. This accumulated knowledge enables insiders to discover the strengths and weaknesses in the defense mechanisms and the systems' structure. Nonetheless, outsiders have little information (in comparison to insiders) about the structure of the systems they attack. Moreover, insiders use legal paths to breach the systems' security throughout legal access, whereas outsiders rely on violating systems security using different methods such as bogus URLs in phishing attacks, SQL injection, Man-in-the Middle attacks … etc. Table 1.1 shows some examples of insiders' attacks and outsiders' attacks according to different criteria [Probst10].

Table 1.1 Insider Attacks vs. Outsider Attacks [Probst10].

Attribute	Outsiders	Insiders
Authentication	Penetrations, attacks on authentication Infrastructures.	Misuse of intended authority by over-authorized users, illegal seizure of super-user access and root keys.
Authorization	Unauthorized exploitation of inadequate controls.	Authorized manipulation of access controls.
Confidentiality	Unencrypted password capture or expose of encrypted passwords.	National security leaks and other disclosures; access to crypto keys.
Integrity	Creating Trojan horses in untrusted components, Word macro viruses, untrustworthy Web code, Man-in-the-Middle attacks.	Inserting Trojan horses or trapdoors in trusted or untrusted components, altering configurations, schedules, and priorities.
Accountability	Masquerading, DoS attacks on accounting infrastructures.	Hacking beneath the audit trails, altering audit logs, compromising misuse detection.

According to different surveys [Gordon05][Cert11], insider threat is as risky as outsiders' threat (hackers) due to the extreme harm that it may pose. The 2005 FBI Computer Crime Survey [Gordon05] reported that trusted insiders were responsible of about 33% of all security breaches in 2005. Similarly, the 2011 Cyber Security Watch Survey [CERT11] showed that 58% of attacks are caused by outsiders, whereas 21% of attacks are caused by insiders. Figure 1.2 shows how the percentage of security breaches by insiders and outsiders have changed over the years, according to the latter survey. Moreover, the survey shows that insider threat is as costly as outsider threat. However, Forrester Research [Forrester11] showed that insider threat is the most costly type of incident. In addition, after analyzing the security practices of more than 300 European, American, and Australian enterprises, Forrester estimated that insiders were responsible for 75% of data security incidents in those enterprises in 2010. Similarly, Verizon Business breach report [Cooper08][Subashini10] stated that outsiders exposed about 30,000 records, whereas insiders exposed about 375,000 records indicating that the cost of insider threat is greatly more than the cost of outsider threat.

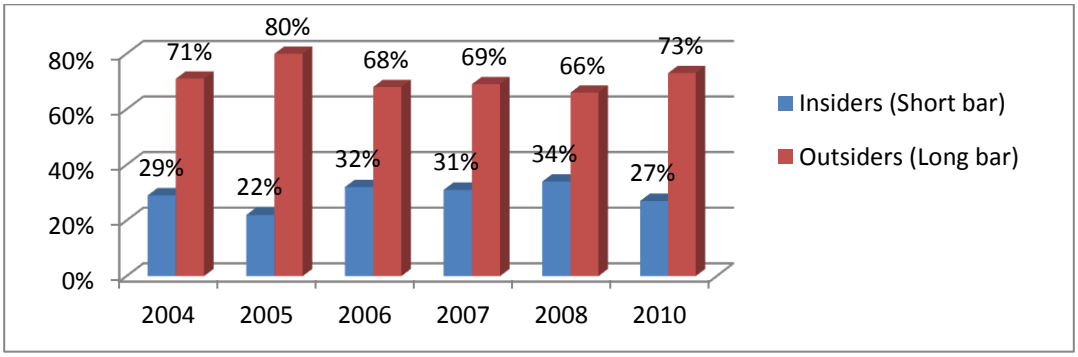

Figure 1.2. Security Violations by Insiders and Outsiders

Obviously, many surveys have shown that insider threat is an immense and urgent security problem. Yet, organizations are investing very little to defend their systems against insider threat. Most organizations' investments are focused on protecting their assets from outsiders' threat.

Organizations rely on insiders' morals and ethics not to violate systems security. Nonetheless, surveys show that this assumption is incorrect. Mechanisms that have been proposed for protecting data from outside attacks are inappropriate to secure systems from authorized users who may misuse their privileges. Thus, the development of mechanisms that protect sensitive data from insiders has become a key demand due to the amount of harm that can be caused by malicious insiders.

1.3 The Contribution of the Dissertation

The focus of this dissertation is on the insider as an object with properties and capabilities that facilitate exposing systems' unauthorized information or making unauthorized changes. The knowledgebases of insiders and their privileges form their power. Therefore, insider threat prediction and prevention mechanisms should identify those properties before going forward in the prediction process. Discovering and representing the knowledgebases and privileges of insiders have gotten a significant focus in this work through developing and using many graphs. As mentioned earlier, the dissertation discusses insider threat in relational database systems in detail. It describes the factors, resources, and features that facilitate insider threat such as dependencies and constraints. In addition, the dissertation demonstrates the paths and approaches that insiders may follow to acquire unauthorized information. Furthermore, it investigates the approaches that insiders may use to make unauthorized modifications to data items. To protect relational databases from these types of insider threat, the dissertation proposes effective methods to detect, prevent, or mitigate those attacks.

In addition, the dissertation shows how different sequences of execution of the same operations of concurrent tasks impose different levels of risk. In addition, it shows how some sequences may lead to limiting the availability of data items or posing threat. For this case, the dissertation shows how to order the operations of concurrent tasks in a safe sequence that prevents insider threat without limiting insiders' tasks. Furthermore, the dissertation investigates the problem in the cloud computing environment and addresses new vulnerabilities that may be used by insiders to launch attacks. It introduces different models to tackle insider threat in cloud relational databases. Finally, the dissertation provides algorithms, theorems, proofs and simulations to prove the efficiency of the proposed approaches in defending relational databases against insider threat.

The rest of the dissertation is organized as follows. The next chapter introduces some related work. Chapter 3 introduces the types of dependencies and constraints in relational database systems. Chapters 4 and 5 discuss the paths and approaches that insiders may follow to acquire unauthorized information or make unauthorized changes in sensitive data items respectively. Moreover, the chapters propose methods to detect and prevent such attacks. Organizing accesses to data items in concurrent tasks is discussed in chapter 6, while chapter 7 discusses insider threat in cloud computing and suggests methods to mitigate insider threat in cloud relational databases. Finally, chapter 8 presents the conclusions.

2. BACKGROUND AND RELATED WORK

Protecting information from insider threat is a very important and difficult research field. Little research has been performed in this area. Moreover, most research in insider threat has been focused at the system level. Very little research has been performed at the application level such as database systems. In this chapter, we introduce some related work that has been accomplished in this area. Furthermore, since dependencies are a major part of insider threat in relational database systems, some work on dependencies and inference channels is introduced. In addition, some work in cloud security is presented to understand the vulnerabilities that threaten cloud relational databases and increase the possibility of insider threat.

2.1 Dependencies and Inference Channels

Dependencies play a major role in the insider threat problem. Dependencies and constraints can be used by insiders to infer unauthorized information or make unauthorized modifications. Dependencies as well as the inference problem have been discussed extensively by many researchers. Most inferences in relational database systems are acquired by combining database constraints with insensitive data items. Inference channels can be discovered during database design [Dawson99, Marks96a, Su91, Yi98] or during queries processing [Marks96b, Stachour90, Farkas07]. In the first approach, database design is modified or the classifications of some data are elevated to remove inference channels. In the second approach, inference is detected during the execution of queries. To remove inference channels in the latter case, malicious queries are either rejected or modified.

Farkas and Jajodia [Farkas07] presented a survey of some research in data inference control in different fields such as statistical and multilevel databases, data mining and web. They discussed how users use insensitive data to get sensitive data to which they have no direct access. Brodsky et al. [Brodsky00] discussed the inference channels that happen when database constraints are used with insensitive data to get information about sensitive data items. They presented the Disclosure Monitor (DiMon) model that detects and removes inference channels that are created by database constraints. Moreover, they used the Disclosure Inference Engine (DiIE) that retrieves all possible information that can be acquired based on users' previous and current queries and database constraints. Their work was well structured and supported by robust algorithms and theorems that analyse the problem and prove the completeness and soundness of the proposed approaches. However, their work considered the problem from a static point of view. That is, they neither considered the updates on data items nor discussed the effects of updates on data items' lifetimes.

Farkas et al. [Farkas01] demonstrated how updates on data items can increase data availability. They stated that updates make data outdated, and stated that the inference based on outdated data does not pose any threat since it would be incorrect. They used this idea to extend the DiMon model used by Brodsky et. al [Brodsky00], which is discussed in the previous paragraph, and developed a new model called Dynamic Disclosure Monitor. The new model ensures that only the inference based on up-to-date data is considered a threat. To facilitate implementing the idea, they established a new mechanism, called the Update Consolidator, that uses a user's history file, updates on data items, and database constraints to generate a new history file for the user in which outdated data are marked, and then, only valid inferences are considered. The authors

assumed that updates always make data expired. Nonetheless, in this dissertation, we show that considering modified data as always expired data may lead to exposing sensitive information. Moreover, we state the conditions that an update process should meet in order to consider an updated data as expired.

Most researchers focus on functional dependencies when dealing with inference channels. However, Yip and Levitt [Yip98] showed that detecting inference channels using functional dependencies only is insufficient. They introduced new inference rules based on analayzing the data stored in databases. The rules are constructed using the overlap between the results of queries, uniqueness of some tuples in databases, and the complement of the results of queries. For instance, as an example of unique tuples, if Bob is the only manager of an age between 30 and 50, then the query that retrieves the salaries of all managers of any age between 30 and 50 exposes Bob's salary. The query itself could be allowed, but the results that it returns in this case should not be allowed. Using the new rules, the authors built an inference detection system that uses a rule based approach to detect inference channels. Morgenstern [Morgenstern87] defined the *INFER* function that computes the amount of knowledge a user can get about data objects in database systems using dependencies and constraints. The INFER function is used to define the set of all information that can be inferred using a core, which is a set of data objects, such as attributes, relationships and inference channels. Morgenstern employed classical information theory, which was developed by Shannon [Shanon48], to measure inference in multilevel databases. In information theory, entropy quantifies the uncertainty of information or the missing information content when the value of a random variable is unknown. The more uncertain or random an event the more entropy it has [Motahari09]. Conditional entropy is the uncertainty of

a random variable when some information is given. Morgenstern used this concept to compute the amount of inference in databases. The idea is based on computing the uncertainty (the range of values) of a data item when no other information is given, and computing the uncertainty of the data item when another data item (or a group of data items) is given. By using these values, the INFER function computes how much information a user can get about a data item by knowing other data items.

2.2 Insider Threat at System Level

Different researchers have introduced different definitions for insiders at system level. Brackney and Anderson [Brackney04] defined an insider as a person who has knowledge of or access privileges to the information or services of a system, whereas they defined a malicious insider as an insider who is motivated to breach a system's security intentionally. Bishop and Gates [Bishop08] defined an insider according to two primitive actions. The first one is to breach a security policy using authorized access, and the second action is to break an access control policy by obtaining unauthorized access. Obviously, the authors defined insiders based on attributes or actions, instead of defining a person as either an insider or not. That is, an insider could be a person or a system that has access privileges to a domain or a system. We should mention here that in this dissertation we use the term "insider" to indicate a malicious insider.

Some researchers used existing methods of detecting external threat to detect insider threat, while others introduced new methods. Spitzner [Spitzner03] used honeypot technologies for insider threat detection. Althebyan and Panda [Althebyan08a][Althebyan08b] introduced new methods to deal with this problem. The authors presented new graphs, which are knowledge

graphs and dependency graphs, that can be used in insider threat prediction and prevention. The knowledge graph of an insider shows the objects about which the insider has information, whereas, a dependency graph shows the dependencies among objects that can be used by insiders to get new knowledge. Bardford and Hu [Bradford05] used intrusion detection mechanisms with forensics tools to detect insider threat in a layered approach.

Park and Giordano [Park06] developed a role-based profile analysis method for preventing insider misuse by focusing on the relationship between insiders and their systems to detect anomalies. Their approach works by analyzing the behaviors of insiders based on their roles. If an insider uses the associated methods in a task according to his/her role, the insider has no malicious intention. However, if the patterns of methods are not appropriate to the task, a malicious activity alert is raised. Hu et. al [Hu06] used the Role Based Access Control (RBAC) and genetic algorithms to generate rules that can detect the differences between users roles and the processes, where the existence of differences indicates insider threat. Chinchani et. al [Chinchani05] proposed a methodology for insider threat assessment, which uses a new threat model called Key Challenge Graph. The new graph relies on an insider's knowledge, the location of the targeted information (key), and the capabilities of the insider to assess the threat. They addressed the conditions of successful attacks and stated formulas for computing the cost of attacks.

Aleman-Meza et. al [Menza05] proposed an ontological approach using semantic associations to detemine the relevance of a document to a domain. Their approach starts by determining the context of investigation, which represents a set of entity classes, relationships, instances and keywords values. The goal of specifying a context of investigation is to capture the types of

entities and relationships that are considered important. Next, documents are processed to produce semantically annontated documents, which indicate documents with metadata that describes them. After that, the relevance of a document to the context of investigation is computed and classified according to a list of relevance levels. Based on the relevance level, the proposed methodology helps in addressing illegal documents access, which in turn detects insider threat.

The research work in this section dealt with insider threat at the system level without considering relational databases. The next section introduces an overview of some research that has been performed on insider threat in relational databases.

2.3 Insider Threat in Relational Database Systems

Very little research has been performed on insider threat at the database level. Chagarlamudi et. al [Chagarlamudi09] used a Petri-Nets model to identify malicious insiders' activities. They used the model to prevent unauthorized modifications in data items. In their work, Petri-Nets are used to model the normal tasks for each user in a system, where places in a Petri-Net represent the transactions of the modeled task. That is, the Petri-Net model of a task represents the partial order of executing the transactions of the task. Any execution of the transactions of a task that does not follow the order in the associated Petri-Net is considered a malicious activity.

Jabbour and Menasce [Jabbour09a] showed a list of security breaches that insiders can launch in systems. In addition, they described a preliminary model that can be integrated into systems for self-protection from insider threat. In their other work [Jabbour09b], Jabbour and Menasce

proposed a self-protection mechanism that is fully integrated into the computing system, called the Insider Threat Security Architecture (ITSA). The new framework forces privileged users, even DBAs, to go through defense mechanisms before making changes to systems or security policies. The authors presented a security scenario which shows that privileged users can expose the system that the users protect, and showed how similar scenarios can be mitigated using the ITSA framework. Other researchers [Hu03][Chung99] developed insider threat detection models based on profiling data access patterns or profiling user access patterns using log files.

Mathew et. al [Mathew10] relied on the results of queries rather than the syntax to detect malicious insiders' activities using a data-centric approach. They claimed that queries with similar syntax can retrieve different results, which enable insiders to launch malicious queries similar (syntactically) to legitimate ones to pose a threat. In order to reduce the complexity of retrieving and checking the possibly huge results of queries, they suggested an approach that picks a small number of tuples that is representative and sufficient to detect insider threat. Garfinkel et al. [Garfinkel02] suggested retrieving a range of results in malicious queries instead of exact results. Using this idea, they provided algorithms for maintaining confidentiality in databases. Their approach adds concealment vectors to database queries which return interval results to protect the original database information from the disclosure of sensitive information. White and Panda [White10] proposed approaches to identify critical data items, which are the target of insiders in general. Addressing critical data items helps in focusing the monitoring process on specific elements of databases which makes fighting insider threat more effective.

Since this dissertation discusses insider threat in cloud Relational Databases, the next section introduces some related work in cloud computing, especially in cloud security.

2.4 Cloud Computing

Cloud computing is a promising technology that offers large-scale and on-demand computing infrastructure. According to Kaufman [Kaufman09], the spending of the US government on cloud computing projects will pass 15 billion dollars by 2015. Achieving low cost live migration is one of the goals of the research on cloud computing. Das et. al [Das11] introduced Albatross, an end-to-end technique for live migration in shared storage databases. Albatross maximizes the availability during a migration process by migrating the cache and the state of active transactions instead of stopping transactions at source nodes and restarting them at destination nodes. Zephyr [Elmore11] minimizes service interruption and increases availability during live migration by using a synchronized *dual mode*. The proposed dual mode enables both the source and destination nodes to execute transactions simultaneously while the migration process is being run. Zephyr transfers the tenant's (migrated application) metadata to the destination to start executing new transactions; meanwhile, the source node continues executing the transactions that were active before starting the migration process.

Cloud security is one of the major problems in cloud computing. Arshad et. al [Arshad09] presented models to quantify cloud security as a set of security metrics. Furthermore, they discussed the problem of random migration of virtual machines. Live migration of a virtual machine (VM) aims to balance the load among all VMs. However, a VM may be migrated to a node without taking into account security requirements. In this case, a VM could be migrated to

a less secure node than the one that is migrated from. The authors called this problem the "random migration problem". They suggested using Service Level Agreements (SLA) by allowing the owner of a VM to determine the security requirements for his/her VM using SLAs; therefore, the resource manager can take into account these requirements before migrating the VM. Wang et. al [Wang09] investigated the problem of data security in cloud data storage. They utilized the homomorphic token with distributed verification of erasure coded data to achieve storage correctness insurance. Hwang et. al [Hwang09] demonstrated a comparison between a number of cloud providers regarding architecture, reliability and security. Furthermore, they addressed outlines for an integrated architecture to guarantee security and privacy in cloud applications. Chow et. al [Chow09] suggested extending control measures in the cloud by using trusted computing and cryptography.

The research in cloud databases is still in its early stages. Few papers have been published in this field. Hacigumus et. al [Hacigumus12] introduced *CloudDB*, a data management platform in the cloud. CloudDB has several features that satisfy the cloud environment. It maintains three types of data stores, which are row store, key-value store, and analytics store, to satisfy different workload types. For instance, analytics store is a read-optimized and a throughput oriented to efficiently handle OLAP workloads, while key-store is used to achieve higher levels of scalability for read/write intensive workloads. Moreover, CloudDB uses both partitioning and replication techniques to achieve availability and scalability. The cloud relational database service has been introduced by some providers such as Amazon [RDS12] (Amazon RDS) and Microsoft [Azure12] (Microsoft SQL Azure). Curino et. al [Curino11] introduced a new comprehensive framework for relational databases on the cloud, called *Relational Cloud*. It

supports new models for efficient multi-tenancy to minimize the resources needed for a workload, an elastic scale-out model to satisfy growing workloads, and models to preserve database privacy. Furthermore, Relational Cloud involves techniques for efficient partitioning, replication, and migration to achieve maximum availability and reliability. Unlike other multi-tenant databases, Relational Cloud does not mix the data of different tenants into a shared database or table. Instead, databases belonging to different tenants are run on the same database server. We should mention here that, to the best of our knowledge, there is no concrete research that has been performed on the problem of insider threat in cloud relational databases.

3. DEPENDENCIES AND CONSTRAINTS

3.1 Introduction

Dependencies and constraints play a crucial role in insider threat in relational database systems (RDBMS). This chapter discusses the types of dependencies and constraints among data items in RDBMS. To demonstrate the problem, let us introduce the following example. Suppose that Figure 3.1 represents a part of the relational database developed by a university. Assume that the database has the following dependencies.

- Rank →Base_Salary.
- {Base_Salary, Experience} → Total_Salary.
- Number of Dependents → HI_Premium.
- {HI_Premium, Total_Salary, Tax} → Net_Salary.
- Score → Grade.

Suppose that Net_Salary and Total_Salary are calculated using the following formulas:

Total_Salary = Base_Salary + 500 * Experience.

Net_salary = Total_Salary – (Total_Salary * Tax + HI_Premium).

The next subsections define the types of dependencies and constraints using this example.

Employee Table

EMP_ID	FName	LName	Rank	HI_Premium	..

Student Table

STD_ID	FName	LName	…

Salary Table

EMP_ID	Base_Salary	Experience	…

Course Table

CRS_NO	Name	Description	…

Dependent Table

EMP_ID	Dependent_Name	Relationship	…

Grade Table

…	Score	Grade	…

Tax Table

Salary	**Tax**	…
<70K	6%	
70K - 90K	8%	
>=90K	10%	

.
.
.

Figure 3.1 A Part of a University Relational Database

3.2 Types of Dependencies

Two data items X and Y have a dependency relationship if at least one of them depends on the other. The dependency between X and Y that is represented by the notation X→Y means that Y depends on X. Dependencies are classified into three types: functional dependencies, multivalued dependencies [Heping05], and fuzzy dependencies [Zuo04]. In addition to these types, the dissertation classifies dependencies according to a number of categories, which are the *strength*, the *direction*, and the *transitivity*. We classify the strength of a dependency into two types: *Strong* and *Weak*, which are defined as follows.

Definition 11 (A Strong Dependency). *Given a dependency X→Y, where X and Y are two data items, the dependency is called a Strong Dependency if a change in X must make a change in Y. This type is represented by X➲Y.*

Definition 12 (A Weak Dependency). *Given the two data items X and Y, if a change in X may or may not make a change in Y, then the two data items have a dependency called a Weak Dependency and is represented by* $X \not\Rightarrow Y$.

For example, the dependency (Rank ➲ Base_Salary), in Figure 3.1, is an example of a Strong Dependency, whereas the dependency (Score ⌦ Grade) is an example of a Weak Dependency.
The *direction* indicates the source (left side) and the destination (right side) of a dependency. The direction of a dependency is classified into *One_Way* and *Two_Way* (Cyclic) dependencies. The following two definitions explain these types.

Definition 13 (A One_Way Dependency). *Given a dependency* $X \rightarrow Y$*, where X and Y are two data items, if Y depends on X but X does not depend on Y, then this dependency is called a One_Way Dependency and is represented by* $X \rightarrowtail Y$.

Definition 14 (A Two_Way Dependency or A Cyclic Dependency). *Given the two data items X and Y and the dependency* $X \rightarrow Y$*, X and Y have a Two_Way Dependency if both data items depend on each other. This type of dependency is represented by* $X \leftrightarrows Y$.

The dependency (Rank ⇸ Base_Salary) is an example of a One_Way dependency. A *Two_Way* Dependency contains two relationships between the two data items, which may be both Strong, both are Weak, or one is Weak and the other is Strong. *Two_Way* Dependencies mostly exist between tables, which may have more than one dependency.

Finally, based on the transitivity property, dependencies are classified into *Direct* and *Transitive* dependencies as follows.

Definition 15 (A Direct Dependency). *Given a dependency X→Y, where X and Y are two data items, the dependency is called a Direct Dependency if a change in X directly affects (make a change in) Y. Such type of dependency is represented by X→ Y.*

Definition 16 (A Transitive Dependency). *Consider the three data items X, Y, and Z that have the dependencies {Y → Z, X →Y}. If a change in X makes a change in Y, and this change in Y makes a change in Z, then Z depends transitively on X and the dependency is represented by X⇢ Z.*

For example, the Dependency (Rank → Base_Salary) is an example of a Direct Dependency, whereas the dependency (Rank ⇢ Net_Salary) is an example of a Transitive Dependency. Obviously, a change in the Rank changes the Base_Salary, which in turn changes the Net_Salary.

Dependencies exist at different levels of granularities in relational database systems. These levels are the *Low Level* (Attribute Level), the *Intermediate Level* (Record level) and the *High Level* (Table Level). All types of dependencies, except the Cyclic (Two_Way) Dependency, are usually found at the Low Level. Actually, some relational database systems may have a Cyclic Dependency at the Low Level, but we have not found a good example of it. Likewise, all types of dependencies exist at the High Level since a table inherits the dependencies that are present at

its Attribute Level. That is, a dependency between two tables is basically a dependency between attributes that belong to the tables. However, two tables may have more than one type of dependency. Similarly, records inherit dependencies from their attributes. This means that various types of dependencies exist at Record Level.

3.3 Constraints on Dependencies

Dependencies may involve constraints. That is, a change on a dependent data item (the right side of a dependency) occurs only when a specified constraint is met on the left side of the dependency. Constraints are classified into two types: those that restrict change in an attribute's value, and those that monitor insertion or deletion of records. Section 3.3.1 discusses how to represent the first type, whereas the second type is discussed section 3.3.2.

3.3.1 Using Petri Nets for Representing Constraints and Dependencies

Representing dependencies and constraints between data items facilitates understanding the relationships between them and the overall structure of relational database systems. For example, in order to change the Grade of a student to 'A', his/her Score should be changed to a value above 90. This is an example about the first type of constraints on the dependency Score → Grade.

The modeling tool that is needed for representation should show the flow of information between different data items at different granularities. Moreover, it should be able to represent various forms of dependencies and constraints. Some constraints could be complex. In some cases, a dependency can be represented using a formula, whereas in other cases it is difficult to represent

dependencies that way. For instance, consider the dependency constraints between the two attributes t_1 and t_2 that says the value of t_2 must equal c_3 when t_1 is in the range c_1 and c_2, and t_2 must equal c_4 when the value of t_1 is less than c_1; otherwise, the value of t_2 is c_5, where c_1, c_2, c_3, c_4 and c_5 are constants.

To meet these goals, the dissertation uses Petri Nets [Murata89] to construct dependency graphs that represents dependencies as well as constraints. Petri Nets are a mathematical and graphical modeling tool. Basically, they are used to represent information processing flow in systems that are nondeterministic, parallel, distributed, asynchronous or concurrent. A Petri Net is a directed, weighted and bipartite graph. It consists of two types of nodes: places and transitions. Places are represented by circles, while transitions are represented by bars or boxes. Arcs are from input places to transitions or from transitions to output places. A weight on an arc represents tokens, which are represented by dots in the input place. A transition can fire if the number of tokens in its input place is greater than or equal to the weight of the corresponding arc. The properties of Petri Nets and other details are beyond the scope of this dissertation. Interested readers may refer to [Murata 89]. Figure 3.2 shows an example of a Petri-Net representing the formula X=2*a+3*b, where the tokens are represented by black spots. In the rest of this dissertation, the conditions for firing are assumed to be met always, thus, the tokens will not be shown.

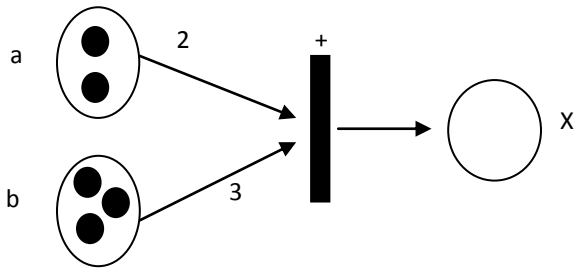

Figure 3.2.Using Petri Nets to Represent a Formula

To show more examples of how Petri-Nets can be used to represent dependencies and constraints, consider the two tables $T_1(t_1,t_2,t_3)$ and $T_2(t_4,t_5,t_6,t_7)$ having the following dependencies and constraints:

- $c_1 =< t_1 <= c_2 \rightarrow t_2 = c_3$
- $c_1 > t_1 \rightarrow t_2 = c_4$
- $t_1 > c_2 \rightarrow t_2 = c_5$
- $t_4 = 3*t_3+1$
- $t_6 = 2*t_2+3*t_5$

The dependency graph that is constructed by using Perti-Nets is called the *Constraint and Dependency Graph (CDG)*. Figure 3.3 shows the CDG of the previous tables using Petri Nets. In the figure, one of the three transitions connected to the attribute t_1 can be fired. Actually, the value of the attribute determines which transition is fired. Thus, the token transfers to one of the constants c_3, c_4 or c_5, which in turn follows its way to the attribute t_2. This makes the attribute t_2 equal the value of the constant from which the token comes. This mapping represents the dependency and constraints between t_1 and t_2. Notice that each attribute name is preceded by the name of the table to which it belongs. The dependency between the attributes $\{t_2, t_5\}$, on one side, and the attribute t_6, on the other side, is represented using the same way of representing the formula in Figure 3.2. Similarly, the dependency between t_3 and t_4 is represented in the same manner. The transition with no inputs is called a source transition. It is used in this graph to make three copies of the attribute t_1. This aims to explain that firing t_6 does not always depend on firing t_1. That is, a change in t_6 may be caused by a change in t_1 or a change in t_2.

Obviously, Figure 3.3 shows how Petri Nets can be used efficiently to construct CDGs. Moreover, since each attribute name is preceded by the corresponding table name, both attribute and table level dependencies and constraints can be represented. Dependencies and constraints between records are shown implicitly. Strictly speaking, a dependency between two records that belong to two different tables can be reduced to a dependency between attributes. For instance, if an attribute *k* in a table B depends on an attribute *j* in a table A, then every record in B depends on its related record in A.

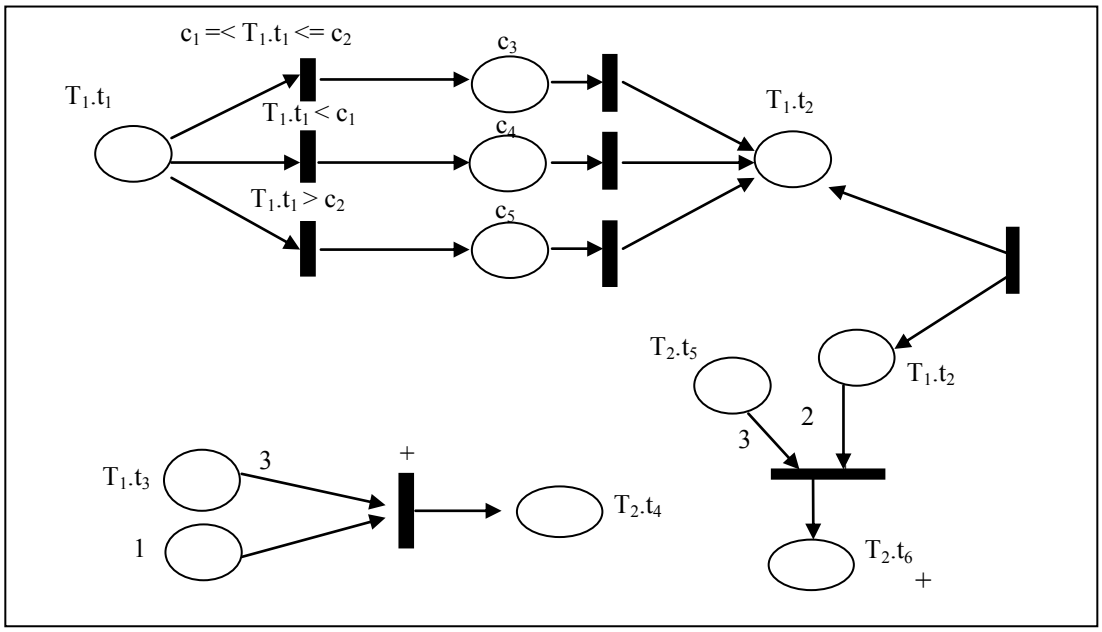

Figure 3.3. A Constraint and Dependency Graph CDG

CDGs are used to build the knowledge graphs of insiders. Strictly speaking, CDGs are used to show how insiders can follow dependencies to infer knowledge about data items to which they do not have access privileges. Constraints in CDGs show what values of data items are stored in the knowledgebase of insiders exactly. The complete details are discussed later in chapter 4.

3.3.2 The Dependency Matrix

Petri Nets can be used to represent constraints on dependencies between attributes. However, it cannot represent the second type of constraints, which is insertion or deletion of records. An insertion/deletion of a record to/from a table may make a change in the dependent table. For instance, consider the dependency between the Employee and the Dependents tables in Figure 3.1. Suppose that the table Employee contains the attribute Health_Insurance_Premium, which depends on the number of dependents of the corresponding employee. In this case, a change in the number of dependents (insertion or deletion of a record into/from the Dependents table) of the employee changes the value of his/her health insurance. Strictly speaking, the insertion or deletion of records changes the related record of the corresponding employee. This type of constraint exists at both table and record levels.

To represent both types of constraints at the table level (and implicitly at the record level), the Dependency Matrix is used. Figure 3.4 represents an example of a dependency matrix that shows dependencies between different tables as well as the constraints on such dependencies. The first column and the first row represent tables. Each cell contains a set of pairs (C, T), where C denotes a constraint and T denotes the type of the dependency. The value 2 means a Strong Dependency and 1 indicates a Weak Dependency. For instance, the cell (T_1, T_2) means that if the constraint C_1 is satisfied on T_1, a change on T_2 must happen since the dependency is strong. As discussed earlier, two tables may have different dependencies, which are represented in the Dependency Matrix by multiple pairs in the given cell. Using the Dependency Matrix, Hot and Safe clusters are constructed. A *Safe Cluster* is defined as follows.

	T_1	T_2	T_3	T_4	T_5	T_6
T_1	-	{(C_1,2), (C_8,2)}	{(C_2,2)}	0	0	0
T_2	0	-	0	{(C_3,2)}	0	0
T_3	0	0	-	0	{(C_4,2)	0
T_4	0	{(C_5,1)}	0	-	0	0
T_5	0	0	{(C_6,2)}	0	-	0
T_6	0	0	0	0	0	-

Figure 3.4. A Dependency Matrix

Definition 17 (Safe Cluster). *Given the Dependency Matrix of a relational database, a Safe Cluster SC ={T_1, … ,T_n} is a group of tables in which each table T_i is independent, directly and transitively, from all other tables that belong to the same cluster.*

Whereas, a *Hot Cluster* is defined as follows.

Definition 18 (Hot Cluster). *Given the dependency matrix of a relational database, a Hot Cluster HC = {T_1,… ,T_n} is a set of tables in which each table T_i is directly dependent on all other tables that belong to the same cluster.*

Based on Figure 3.4, Figure 3.5 shows examples of Hot and Safe Clusters, where Hot Clusters are represented by dashed ovals and Safe Clusters are represented by solid ones. For instance, clusters C_3 to C_7 are Safe Clusters, whereas clusters C_1 and C_2 are Hot Clusters. As shown in Figure 3.5, Hot Clusters and/or Safe Clusters may overlap. Notice that tables that belong to different clusters may still have a dependency relationship, but not a Cyclic Dependency. For instance, tables T_1 and T_2 still have a One_Way dependency.

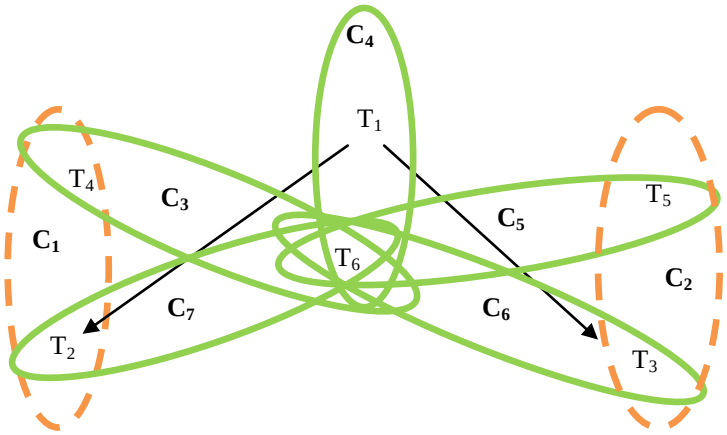

Figure 3.5. Hot and safe clusters

4. INSIDER THREAT: UNAUTHORIZED KNOWLEDGE AQUISTION

4.1 Introduction

Insiders may be able to predict the values of data items, which they may not be authorized to access, by investigating dependencies and constraints. For instance, in Figure 3.3 in Chapter 3, assume that an insider has a read access to t_1 and has no authorized access to t_2. In this case, s/he can infer the value of t_2 using the associated dependencies and constraints. Similarly, assume that the insider has a read access to t_2 and has no access to t_1. Then, if the value of t_2 is changed to c_3, the insider would realize that the value of t_1 has been changed to c_1. The latter case is called *Cyclic Inference*. Likewise, the insider can deduce the correct value of t_1 if the value of t_2 is changed to either c_4 or c_5. Thus, insiders may use their knowledge about data dependencies and constraints to acquire knowledge about some data items to which s/he has no authorized read access. This chapter demonstrates how insiders can get unauthorized information and broaden their knowledgebases using dependencies and constraints.

4.2 Insiders' Knowledge

Understanding the various types of dependencies and constraints in relational databases enables us to discover the knowledge that insiders can get. As discussed earlier, insiders may use dependencies and constraints to acquire unauthorized knowledge, which are classified into three types: *inferred knowledge*, *computed knowledge*, and *aggregated knowledge*. The next subsections address how insiders can acquire these different types of knowledge.

4.2.1 Inferred Knowledge

The type of knowledge that insiders can get using inference is called *inferred knowledge*, which is defined formally as follows:

Definition 19 (Inferred Knowledge). *Given a dependency A → B in a relational database, where A and B are data items, the knowledge that an insider deduces about B by accessing A is called inferred knowledge.*

The knowledge that an insider infers can be partial or exact. To measure the amount of knowledge an insider can get, the INFER function [Morgenstern87] is used, which is as follows:

$$INFER\ (x \rightarrow y) = (H(y) - H_x(y))\ /H(y) \qquad (1)$$

INFER ($x \rightarrow y$) represents the amount of information about a data item *y* that can be inferred using a data item *x*. $H(y)$ represents the uncertainty of *y* and $H_x(y)$ represents the uncertainty of *y* given *x*. The amount of inferred knowledge ranges between 0 for no knowledge to 1 for exact knowledge.

An insider may use functional dependencies to infer knowledge. For instance, let us consider the trivial example of the functional dependency between the attribute "Score" and the attribute "Grade" (Score → Grade). Figure 4.1 shows the dependency and constraints. Hence, the dependency is a Weak and a Direct dependency.

Score	Grade
>=90 and <=100	A
>=80 and <90	B
>=70 and <80	C
>=60 and <70	D
<60	F

Figure 4.1 A Functional Dependency

An insider can infer information if he/she has access to any side of the dependency, but the amount of information s/he can get differs. For instance, the uncertainty of the Grade is 5 since we assume that grades range from A to F. However, the uncertainty of the Grade given the Score is 0 since if the insider is given a Score, he/she can directly and exactly infer the Grade value. The amount of information the insider can get about the Grade is calculated as follows:

$$\text{INFER}(\text{Score} \rightarrow \text{Grade}) = (H(\text{Grade}) - H_{\text{Score}}(\text{Grade})) / H(\text{Grade}) = (5-0)/5 = 100\%.$$

This means that the insider can get exact knowledge about the Grade if he/she has an access to the Score. On the other hand, if the insider has an access to the Grade attribute only, the knowledge that s/he can get is partial since the dependency is Weak. Moreover, the amount of information s/he can get using this dependency is either 90% or 40% since the Score intervals are not equally likely. Notice that the insider can infer information using Cyclic Inference. Hence, if the dependency is a Strong, the insider can get exact knowledge.

The amount of information that can be inferred when the Score interval is 10 is computed as follows:

$$\text{INFER}(\text{Grade} \rightarrow \text{Score}) = (\text{Score} - H_{\text{Grade}}(\text{Score})) / H(\text{Score}) = (101-10)/101 = 90\%.$$

While the amount of information that can be inferred when the Score interval is less than 60 is computed as follows:

$$\text{INFER(Grade} \rightarrow \text{Score)} = H(\text{Score}) - H_{\text{Grade}}(\text{Score})) / H(\text{Score}) = (101 - 60)/101 = 40\%.$$

An insider can also gain knowledge using fuzzy dependencies. For example, consider the fuzzy dependency between the "Project type" attribute and the "Project name" attribute as shown in Figure 2. Suppose that James is working at the company F as a programmer, then the uncertainty about which projects he/she is working on is 4 since there are 4 projects at the company. However, if the insider is given that James is working on an accounting project (given access to project type), then the amount of knowledge the insider can get about the projects on which James is working is computed as follows:

$$\text{INFER(project type} \rightarrow \text{project name)} = (\text{project name}) - H_{\text{project type}}(\text{project name}))/H(\text{project name}) = (4-2)/4 = 50\%.$$

Project type	Project name	Due date
Accounting	A	Jan,1
Accounting	B	Feb,1
Marketing	C	Jan,1
Marketing	D	Feb,1

Figure 4.2 Projects in a Company F

Multivalued dependencies [Su87] do not enable insiders to infer information. To clarify this point, consider Figure 4.3 that shows an example of a trivial multivalued dependency between the attribute Emp and the attribute Project. Suppose that an insider has an access to the records of employee B, and the insider has no access to the records of employee A. Using the records of B,

the insider can get the values of the "Project" attribute of employee A, which are M and N in the example.

	Emp	Project
t_1	A	M
t_2	A	N
t_3	B	M
t_4	B	N

Figure 4.3 Multivalued Dependency

However, the following observations need to be taken into consideration in the case of multivalued dependencies. If the insider is familiar with the multivalued dependencies as well as the constraints, accessing some records in a table that has a multivalued dependency does not provide any new information to the insider. Meanwhile, if the insider is not familiar with the constraints of the multivalued dependency, the access of the insider to some records gives him/her exact knowledge about other records in that table. For instance, if an insider knows that every employee is working on the projects M and N, as shown in Figure 4.3, accessing the records of an employee does not reveal any new information about other employees. Since we assume that insiders are familiar with the dependencies as well as the related constraints, we will not consider this case as an inference problem.

An insider can also use a combination of accessed attributes to infer information. That is, the variable *X* in the INFER function could be a set of attributes. In this case, the insider can use all attributes in *X* together to infer information about *Y*. The modified version of the INFER function [Morgenstern87] is:

$$INFER (\{x_1,..,x_n\} \rightarrow y)=(H(y)-Hx_1,..,x_n(y))/H(y) \qquad (2)$$

For instance, in the example of Figure 4.3, suppose that an insider has access to the "Due date" attribute in addition to the "Project type" attribute. Then the amount of information that can be inferred is computed as following:

$$\text{INFER}(\{\text{project type, due date}\} \rightarrow \text{project name}) = (H(\text{project\_name}) - H_{\text{project-type,due-date}}(\text{project\_name}) / H(\text{project\_ name}) = 100\%.$$

The inference that has been discussed so far is direct inference. However, insiders can infer knowledge transitively using transitive dependencies (Definition 7), which is called *transitive inference* and is defined as follows:

Definition 20 (Transitive Inference). *Given a transitive dependency $A \rightarrow B \rightarrow C$ in a relational database D, where A, B, and C are data items in D, the knowledge an insider infers about C by accessing A is called transitive inference.*

4.2.2 Computed Knowledge

Computed knowledge is similar to inferred knowledge except that it is acquired through computation. Formally, the computed knowledge is defined as follows:

Definition 21 (Computed Knowledge). *Given a dependency $A \rightarrow B$ in a relational database, where A and B are data items, the knowledge an insider gets about B through computation by using A is called computed knowledge.*

For example, consider the dependency between the Rank attribute and the Total_Salary attribute in Figure 4.4. Suppose that the range of total salaries of academic staff is between 65k and 130k,

and there are three ranks for academic staff, which are known to insiders. In addition, suppose that the Total_Salary of an academic staff is computed as follows: Total_Salary = Base_Salary + 500 * experience, where Base_Salary is the left side of the ranges of Total_Salary. If an insider has an access to the Rank attribute, the amount of information he/she can infer about the Total_Salary of any academic staff is either 80 % or 61.5% because Total_Salary intervals are not equally likely. If the insider has an access to the Rank attribute and the Experience of an academic staff, the information he/she can acquire about the Total_Salary attribute is 100% (exact knowledge), which is computed as shown.

$$INFER(\{Rank, Experience\} \rightarrow Total\_Salary) = (H(Total\_Salary) - H_{rank,experience}(Total\_Salary)) / H(Total\_Salary) = 100\%.$$

Rank	Total_Salary
Assistant Prof	65k-90k
Associate Prof	91k-116k
Full Prof	117k-130k

Figure 4.4. Rank → Total_Salary Dependency

The above computation is based on the assumption that the left side of a Total_Salary interval is the Base_Salary of an academic staff without any experience, and the right side is the maximum total salary an academic staff (with respect to the corresponding Rank) can get.

4.2.3 Aggregated knowledge

Aggregated knowledge is the knowledge that results from combining two or more data items. It may be called *composite knowledge* or *combined knowledge* as well. Formally, the aggregated knowledge is defined as follows:

Definition 22 (Aggregated Knowledge). *Given two related data items A and B in a relational database, the knowledge achieved by combing A and B together is called aggregated knowledge.*

An insider may have direct access to basic knowledge units (attributes or virtual knowledge units) or aggregated knowledge units. To show how aggregated knowledge could be a threat, consider the following example. Suppose that the relation R(Name, Rank, Salary) is a relational schema and the knowledge units Name, Rank and Salary can be accessed separately by an insider. Similarly, the aggregated knowledge [Name, Rank] or [Rank, Salary] can be accessed separately by the insider, but the insider should not get access to the aggregated knowledge [Name, Salary], which is unauthorized information to the insider. Obviously, using the two aggregated knowledge [Name, Rank] and [Rank, Salary] and the dependency between the two attributes Rank and Salary, the insider obtains the aggregated knowledge [Name, Salary], which is considered a threat [Brodsky00].

Figure 4.5 shows an example of how aggregated knowledge is constructed, where KU indicates a knowledge unit and AK indicates an aggregated knowledge unit. Aggregated knowledge can be gained using dependencies and the transitivity property. Therefore, in case of the dependency

A→B→C, if an insider has access to data item A, he/she can get the aggregated knowledge [A,B], [B,C] and [A,C]. Formally, an insider *R* can aggregate knowledge as follows.

$$(A \rightarrow B \rightarrow C) \wedge Access(R,A) \rightarrow Access(R,[A,B]) \wedge Access(R,[B,C]) \wedge Access(R,[A,C])$$

Where Access (R, [A,B]) means that an insider *R* has access to the aggregated knowledge [A,B].

The knowledge that can be acquired by an insider, but not stored in the database, is called *virtual knowledge* [Morgenstern87]. Thus, an insider can have two types of knowledge: *stored knowledge*, which is stored in the database, and virtual knowledge.

The term "data item" may represent an attribute, a record, or a table. Dependencies between tables or records occur due to dependencies between attributes in those tables or records. Thus, in this work, the discussion about knowledge or a dependency relationship at a lower level of granularity (attribute level) is applicable for other levels to granularity. However, in the next sections, we use the term "data item" to indicate a basic knowledge unit (a stored basic knowledge unit, which is an attribute, or a virtual basic knowledge unit).

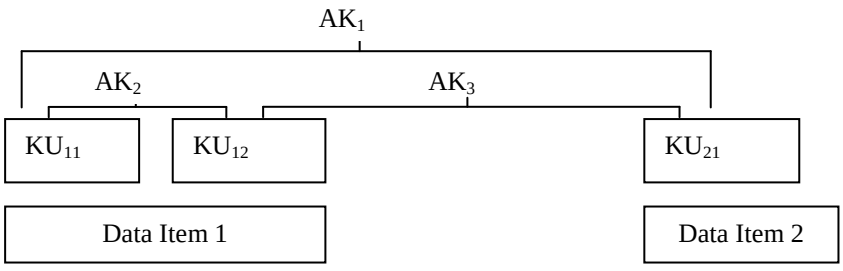

Figure 4.5 Aggregated Knowledge

Based on the concepts of knowledge acquisition and Figure 3.1, Figure 4.6 demonstrates how an insider broadens his/her knowledge using his/her access privileges, and dependencies and constraints.

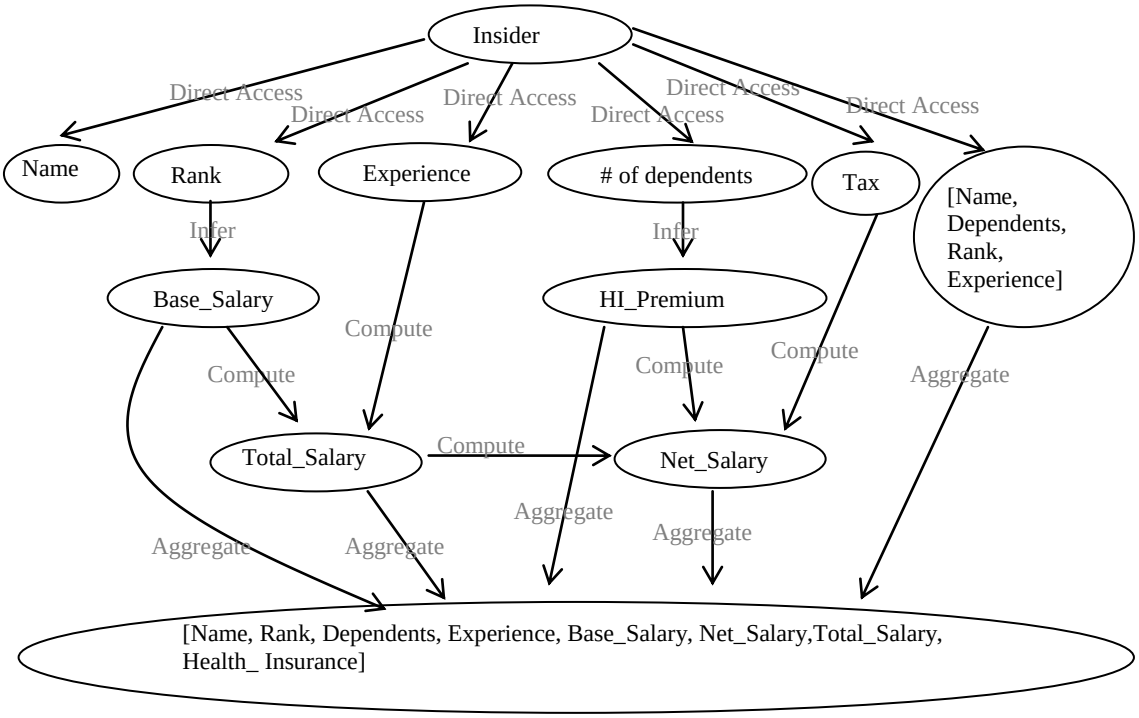

Figure 4.6 Acquiring knowledge

4.3 Neural Dependency and Inference Graph

A dependency graph can be used to show dependencies among different data items in a relational database system [Althebyan07]. In addition to CDG, this dissertation introduces another type of dependency graphs called the *Neural Dependency and Inference Graph (NDIG)*. An NDIG represents dependencies among data items in relational databases, the amount of knowledge that can be acquired from/by accessing a single data item about other data items, and the amount of

knowledge that can be deduced from/by accessing a group of data items about other data items. The NDIG is defined formally as follows:

Definition 23 (NDIG). *The Neural Dependency and Inference Graph NDIG (O,N,W,E) is a graph that shows dependencies among data items and the amount of information that can be acquired about data items using dependencies, where:*

1. *O represents data items, which are demonstrated by rectangles.*
2. *N indicates neurons, which are represented by ellipses.*
3. *W indicates weights on edges.*
4. *E indicates edges, which represent dependencies among data items such that:*
   a. *The edges $E(O_i, N_k)$ and $E(N_k, O_j)$ indicate that the data item O_j depends on the data item O_i.*
   b. *The weight on the edge $E(O_i, N_k)$ represents the amount of information that can be acquired about the data item O_j using the data item O_i.*
   c. *The weight on the edge $E(N_k, O_j)$ represents the amount of information that can be acquired about the data item O_j using all data items $\{O_1 \ldots O_x\}$ together, such that $\forall_{s=1}^{x} E(O_s, N_k) \in E$.*

An example of NDIG is shown in Figure 4.7. NDIG uses some ideas from artificial neural networks, where rectangles represent data items (input or output data items), elliptical nodes represent neurons, and weights on edges represent inputs to neurons or output from neurons. Each neuron consists of the function INFER and a weight. The weight, which is the output, represents the amount of information that can be inferred about the output data items using the input data items. Obviously, the INFER function is used to compute weights in neurons. Solid

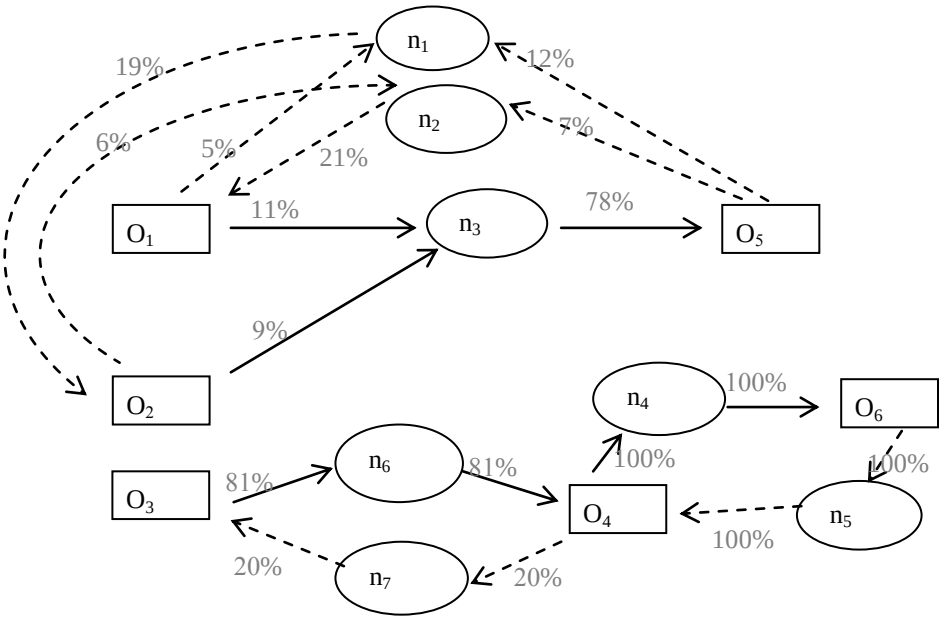

Figure 4.7 Neural Dependency and Inference Graph (NDIG)

lines show that there is a dependency between input and output data items, which indicates that output data items depend on input data items. Dashed lines represent cyclic inference, which was discussed earlier. For instance, as shown in Figure 4.7, O_5 depends on O_1 and O_2. The weight 11% on the edge e(O_1, n_3) means that an insider who accesses O_1 can acquire about 11% of knowledge about O_5. Furthermore, the weight 78% on the edge e(n_3 ,O_5) means that an insider who accesses both O_1 and O_2 can infer about 78% of knowledge about O_5.

Figure 4.8 demonstrates the NDIG of the academic staff database that is shown in Figure 3.1. As shown, the amount of knowledge that can be acquired about the Net_Salary using the Tax table only is 0%. As assumed, some insiders may already know dependencies and constraints. Thus, accessing the Tax table does not reduce the uncertainty of the Net_ Salary for them. Similarly, using the HI_Premium (Health_Insurance_Premium) value only provides negligible information

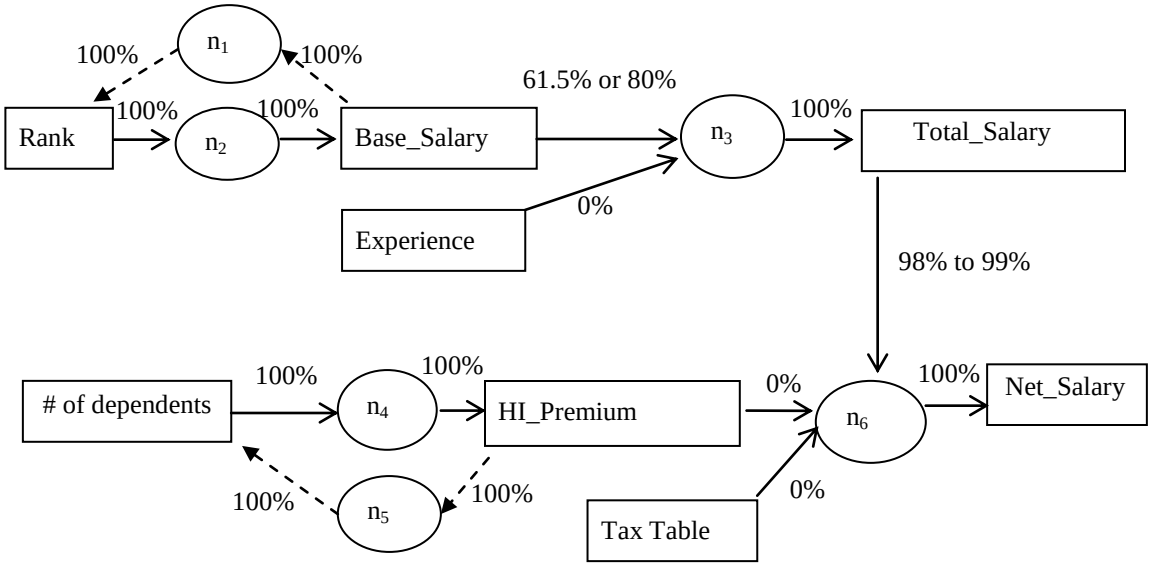

Figure 4.8 NDIG for the Academic Staff Database

since it does not reduce the uncertainty of the Net_Salary. To clarify this point, suppose that possible minimum and maximum values of Total_Salary based on Figure 4.4 are $65000 and $130000 respectively. The range of Net_Salary without knowing HI_Premium is [65000 - Tax, 130000 - Tax]. The range of values of Net_Salary given HI_Premium is [65000 - HI_Premium - Tax, 13000 – HI_Premium - Tax], which does not reduce the uncertainty of Net_Salary. Thus, the edge that connects both knowledge units is labeled with 0% as shown in Figure 4.8.

However, the insider who accesses the Total_Salary of an employee gets a huge amount of information about the Net_Salary of the employee. Obviously, the insider can use Total_Salary in conjunction with his/her knowledge about Tax table to get information about the Net_Salary of the employee. Strictly speaking, he/she can get a reduced range of the Net_Salary of the employee. Using these criteria, the amount of information the insider can acquire about the Net_Salary ranges between 98% and 99%, which is based on the assumption that the maximum

HI_Premium is 1000. However, the insider can acquire the exact knowledge about the Net_Salary of an employee if he/she knows exactly the HI_Premium and the Total_Salary values. To make the NDIG in Figure 4.7 simpler, we have omitted most cyclic inference lines.

4.4 Insiders' Knowledgebases

A knowledgebase determines which data items the corresponding insider has read. That is, it is a profile of insider accesses to data items. This section demonstrates how to build up the knowledgebases of insiders at different levels of granularities. The dissertation defines knowledgebase as follows.

Definition 24 (Knowledgebase). *Given an insider R and a relational database RDB, the knowledgebase of R, written as KB(R), is the set of data items in RDB that R has accessed using his/her privileges, and the data items about which R can acquire information using dependencies and constraints.*

Based on the concepts of Hot and Safe Clusters in section 3.2, the following observations are made. First, if an insider gets read access to a table that belongs to a Hot Cluster, s/he can acquire information about all other tables in that cluster. Secondly, if an insider gets read access to a table in a Safe Cluster, s/he cannot infer any information about any other table that belongs to the same cluster (without accessing them directly). Finally, if an insider gets read access to a table in some cluster, s/he still may infer information about other dependent tables that belong to different clusters. The latter case would occur when the dependency between them is One-Way dependency.

In addition to the above conclusions, an insider can acquire information transitively about other

tables using the transitive dependencies among attributes as follows. Suppose that a relational database has the following dependency: $T_1 \rightarrow T_2 \ldots T_{n-1} \rightarrow T_n$, where T_1 to T_n are tables in the database. The insider who has a read access to T_k ($1 \leq k \leq n\text{-}2$) can infer information about T_j transitively (transitive inference), where j ranges from k+2 to n, if the dependencies between the tables T_k to T_j are between attributes in the form $C_k \rightarrow C_{k+1} \rightarrow C_{k+2} \ldots C_{n-2} \rightarrow C_{n-1} \rightarrow C_n$ such that $\forall_{x=k}^{n} C_x \in T_x$. Obviously, a transitive dependency is formed by a sequence of connected direct dependencies. Notice that the condition imposes the continuity of the dependencies between tables. Strictly speaking, the destination attribute (right side) in a direct dependency is the source attribute (left side) for the next direct dependency. Hence, the insider who has access to T_n can infer information about predecessors in the chain in the same way (cyclic inference).

The existence of inferable tables in the knowledgebase of an insider does not necessarily mean that the insider can infer all information about those tables. To reveal more details, dependencies and constraints between attributes in those tables should be investigated. To perform this, the dependency graph CDG is used. For instance, in Figure 3.3, suppose that an insider has full read and write access on table T_1. Then, both tables T_1 and T_2 are added to the knowledgebase of the insider since they have a dependency relationship. The insider knows all information about T_1, whereas his/her information about T_2 is limited by the dependency between the two tables. To clarify what information the insider can infer about T_2, dependencies between attributes in both tables should be investigated. Clearly, s/he infers information about a_4 and a_6, and acquires information about a_5 by cyclic inference. However, s/he does not have information about other attributes in T_2. To compute how much information the insider has about specific attributes, the corresponding NDIG is used.

A similar scenario is used at the records level. If the insider has read a record, then the record is added to his/her knowledgebase. In addition, other records that depend on this record are added to. But this does not mean that s/he has full information about the latter records. To determine what information the insider has about those records, the dependencies among attributes should be investigated.

4.4.1 Knowledgebase Algorithm

Algorithm 4.1 shows how to build a knowledge graph, which represents knowledgebases at different levels of granularity. It uses the NDIG and the CDG of the corresponding relational database as well as the Dependency Matrix. In addition, it uses Hot and Safe clusters to facilitate construction of the knowledge graph.

The algorithm starts by adding the insider as the root of a knowledge graph. The second level of the graph contains the tables about which the insider has information (by direct access or by inference). For each table at the second level, the algorithm determines which attributes the insider has information of (by direct access or by inference). The NDIG is used to label edges by the amount of information the insider can have about each data item (attribute or table). Either the NDIG or the CDG is used to show dependencies between knowledge units (attributes), where dependencies are represented by an edge (arrow) from the source attribute (left side of a dependency) to the destination (dependent) attribute. Moreover, the CDG is used to show what values of attributes are stored in the insider's knowledgebase, which is used in insider threat prediction and prevention later in section 4.5. Notice that the amount of information the insider

has about a table is the average of all information s/he has about all attributes that belong to the table.

Algorithm 4.1. Knowledgebase Algorithm

Input: An insider I, Dependency Matrix, CDG, NDIG, Hot and Safe clusters, S: Set of tables to which the insider has direct read access.

Output: The knowledge graph of the insider I.

1. Initialize the KG = (V,E), where V = {I}, E={}.
2. **For** each table T_k in S //*add directly accessed tables*
3. V=V ∪ T_k // *add the node T_k to KG*
4. E=E ∪ {e(I, T_k)} // *add the edge e(I, T_k) to the KG*
5. **For** each t ∈ attributes(T_k) and the insider has a read access to it // *add directly accessed attributes*
6. V=V ∪ {t} // *add the attribute t to KG*
7. E=E ∪ {e(T_k, t)} // *add edge e(T_k, t) to the KG*
8. **Endfor**
9. **Endfor**
10. **For** each T_k in S do // *consider dependencies*
11. **For** each Safe Cluster R to which T_k belongs
12. $\forall X \in R \wedge X \notin D \rightarrow X \notin KB(I)$ //*exclude X from KG*
13. **Endfor**
14. **For** each Hot Cluster H to which T_k belongs
15. **For** $\forall T_m \in H \wedge T_m \neq T_k$
16. V = V ∪ {T_m}// *add the node T_m to KG*
17. E = E ∪ {e(I, T_m)} // *add edge e(I, T_m) to the KG*
18. **For** each t_m ∈ attributes(T_m) $\wedge$ $t_k \rightarrow t_m$, where t_k ∈ attributes(T_k) // *add directly inferred attribute(s) to the KG*

19. V = V ∪ { t_m } // *add the attribute t_m to KG*

20. E = E ∪ { e(T_m, t_m)} // *add e(T_m, t_m) to KG*

21. E = E ∪ { e(t_k, t_m) } // *add e(t_k, t_m) to KG*

22. **Endfor**

23. **Endfor**

24. **Endfor**

25. **For** each other table T_s that has dependency (one_way) with T_k // *add tables from other clusters*

26. Repeat steps 16 to 22 for the table T_s

27. **Endfor**

28. **For** each table T_j that depends transitively on T_k (Definition 20) // *transitive inference*

29. V=V ∪ { T_j } //*add the node T_s to KG*

30. E=E ∪ {e(I, T_j)} // *add edge e(I, T_j) to the KG*

31. **For** each $t_j \in$ attributes(T_j) $\wedge$ $t_k \dashrightarrow t_j$ (transitive inference), where $t_k \in$ attributes(T_k) //*add the transitively inferred attribute(s) to the KG*

32. V=V ∪ { t_j } // *add the attribute t_j to KG*

33. E=E ∪ {e(T_j, t_j)} // *add edge e(T_j, t_j) to the KG*

34. E= E ∪ {e(t_k, t_j)} // *add edge e(t_k, t_j) to the KG*

35. **Endfor**

36. **Endfor**

37. **Endfor**

38. **For** each edge e(T,t) // *T is a table and t is an Attribute*

39. Weight (e(T,t)) = the amount of information the insider has about t // *using NDIG*

40. **Endfor**

41. **For** each edge e(I,T) // *weight of tables*

42. Weight(e(I,T)) = $\sum_{i=1}^{n} Weight(e(T, t_i))\ /n$, where n is the number of attributes in T_z.

43. **Endfor**

To clarify how this algorithm works, suppose that the corresponding NDIG for the CDG in Figure 3.3 is as shown in Figure 4.9.

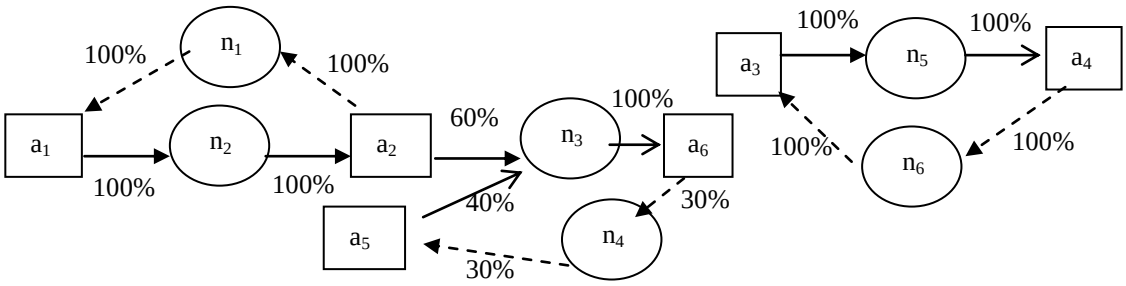

Figure 4.9. The NDIG of the Database in Figure 3.3

Let the insider have read T_1. In this case, Figure 4.10 shows the KG of the insider based on the algorithm. Solid arrows point to data items to which the insider has direct access, whereas dotted arrows point to data items about which the insider can infer information. Dashed arrows represent the paths the insider follows to infer information. For instance, the insider acquires information about the attribute a_2 using direct access, whereas s/he gets information about a_6 by inference using the attribute a_2. Weights on edges show the amount of information the insider can have about the destination data items. The weight on edges between the root and a table is the average value of weights on the edges between the table and its corresponding attributes. Notice that these values are based on the assumption that the tables do not have any other attributes other than what are shown.

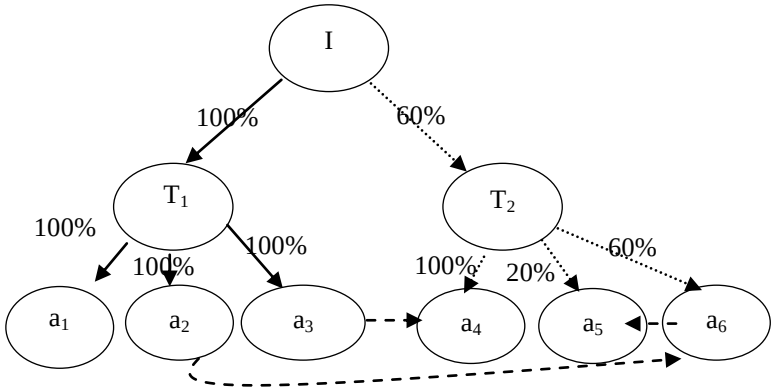

Figure 4.10. The Knowledge Graph KG of an Insider

4.4.2 Proof of Correctness of Algorithm 4.1

Insiders can get knowledge in different ways. First, they can get knowledge directly using their privileges. Second, they can infer knowledge directly (using direct dependencies). Finally, they can acquire knowledge transitively (using transitive dependencies). The following theorems prove that the algorithm considers all these ways when building knowledge graphs.

Theorem 1. *Given a table B in a relational database DB, and the knowledge graph KG of an insider I,*

$$B \in KG \Leftrightarrow B \in D(I) \vee B \in DD(A) \vee B \in TD(A)$$

where D(I) is the set of tables to which the insider has direct access, DD(A) and TD(A) are the set of tables that depend directly or transitively on a table A respectively, and $A \in D(I)$.

The theorem lists the three ways which insiders follow to get knowledge. The following proof proves that the algorithm adds every table about which the insider may get knowledge using those ways.

Proof:

(→) Suppose that A ∈ D (I), and DB has the following dependencies:

- A→B (direct dependency).

- A ⇢ C (transitive dependency).

It is obvious that A ∈ KG by steps 2 to 4, which add every directly accessed table to the knowledge graph. This proves the part of directly accessed tables. In the case of table B, let H(A) be a Hot Cluster that contains A. Now, since A → B, then either B ∈ H(A) (A and B have a cyclic dependency), or B ∉ H(A). In the first case, B ∈ KG by steps 14 to 17. In the second case, B ∈ KG by steps 25 to 27, which proves the part of directly dependent tables. In the case of C, where C depends transitively on A, C ∈ KG by steps 28 to 30, which proves the part of transitively dependent tables. In summary, all tables about which the insider can get knowledge are added to the knowledge graph. □

(←) (Proof by Contradiction) This part proves that no other table is added to the knowledge graph of the insider. Suppose that ∃B ∈ KG: (B ∉ D(I) ∧ B ∉ DD(A) ∧ B ∉ TD(A)), where A ∈ D(I). In this case, B should exist in a Safe Cluster. Thus, B ∉ KG by steps 11 to 13, which is a contradiction. □

As discussed earlier, the existence of a table in the knowledge graph of an insider does not mean that the insider has knowledge about every attribute in that table. The following theorem clarifies this point.

Theorem 2. *Given an attribute k, where k ∈ B for some table B in a relational database DB, and the knowledge graph KG of an insider I:*

$$k \in KG \Leftrightarrow (k \in DA(I) \vee k \in DDA(s) \vee k \in TDA(s))$$

where DA(I) is the set of attributes to which the insider has direct access, DDA(s) and TDA(s) are the set of attributes that depends directly or transitively on an attribute s respectively, and s ∈ DA(I).

The theorem states that an attribute belongs to the KG of an insider if and only if it is accessed directly, inferred directly or inferred transitively by the insider. The following proof proves the correctness of Theorem 2.

Proof:

(→) Suppose that the DB has the following dependencies:

- A → B ... →T, where A, B and T, etc. are tables in DB.
- k → r ... → z, where k, r and z etc. are attributes such that k ∈ A, r ∈ B and z ∈ T , and k ∈ DA(I) and A ∈ D(I).

First, steps 5 to 8 state that ∀ k: (k ∈ DA(I) : (k ∈ A ∧ A ∈ D(I))) → k ∈ KG. This proves the part of directly accessed attributes. Second, steps 18 to 22 state that ∀ r: (r ∈ DDA(k) : (k ∈ A ∧ A ∈ D(I) ∧ r ∈ B ∧ B ∈ DD(A))) → r ∈ KG. This proves the part of directly dependent attributes. Finally, steps 31 to 35 state that ∀ z: (z ∈ TDA(k) : (k ∈ A ∧ A ∈ D(I) ∧ z ∈ T ∧ T ∈ TD(A))) → z ∈ KG. This proves the third case. □

(←) (Proof by Contradiction) Suppose that ∃k ∈ KG: (k ∉ DA(I) ∧ k ∉ DDA(s) ∧ k ∉ TDA(s)), where s ∈ DA (I). In this case, there are four cases:

1. k ∈ A , where A ∈ D(I) but k ∉ DA(I). In this case, k is excluded (not added) using steps 5 to 8.
2. k ∈ B , where B ∈ DD(A) but k ∉ DDA(s) for some A ∈ D(I) and s ∈ DA (I). In this case, k is excluded using steps 18 to 22.
3. k ∈ T , where T ∈ TD(A) but k ∉ TDA(s) for some A ∈ D(I) and s ∈ DA(I). In this case, k is excluded using steps 31 to 35.
4. k ∈ A and A ∈ S(P), where S(P) is a Safe Cluster of a table P about which the insider has knowledge. In this case, k is excluded using steps 11 to 13.

Obviously, all mentioned cases contradict the assumption. □

4.5 Insider Threat Prediction and Prevention

As discussed earlier in previous sections, insiders can use their read access privileges, dependencies and constraints to acquire information about unauthorized data items. In addition, data items in a knowledgebase could be risky. This section introduces the role of a knowledgebase and the life times of data items in insider threat situation. Moreover, it introduces the proposed models for insider threat prediction and prevention.

4.5.1 The Role of Knowledgebase and Lifetimes of Data items in Insider Threat

The values of data items in the knowledgebase of an insider may be combined with some

insensitive data items that the insider may request to infer sensitive information, which poses a threat [Yaseen09][Yaseen10b]. Revoking read accesses from previously accessed data items does not eliminate the threat since the values still exist in the insider's knowledgebase. For instance, consider the dependency ({Rank, Experience} → Total_Salary). If the insider has accessed the Rank attribute (which is added to his/her knowledgebase) and then he/she is given a read access to the Experience attribute, he/she can combine both data items to infer the value of the Total_Salary attribute, which could be sensitive information.

Clearly, an insider's knowledgebase could pose a serious threat, but not if the data items in the knowledgebase are expired. That is, if other insiders modify the data items, the lifetimes of those data items (old values) may expire. Thus, using them to infer sensitive information would not pose a threat. In light of this, considering the lifetime of data items in an insider's knowledgebase is important. However, merely updating values of data items does not always make their lifetimes expire. Changing the value of an attribute that belongs to a Strong Dependency makes it expire. However, it may or may not expire if the attribute belongs to a Weak Dependency. To clarify this point, consider the Strong Dependency (Rank → Base_Salary). In addition, assume that an insider, say K, has the information (Jiff, Assistant Professor) in his/her knowledgebase about the professor Jiff. In this case, K can infer exact information about the Base_Salary of Jiff. However, changing the Rank attribute of Jiff by other insiders must change the Base_Salary of Jiff. In this case, the Rank value in K's knowledgebase is expired. That is, if K uses it to infer the Base_Salary of Jiff, K's inference will be incorrect.

On the other hand, consider the Weak Dependency (Score → Grade) in a student table. A change

in the Score attribute does not always make a change in the Grade attribute. That is, changing Score does not always make the old value of Score expire. To clarify this point, suppose that the insider has read a student's score, say 85, which enables him/her to infer the student's grade (B in this case) . However, suppose that the student's score has been updated to 88 and the insider is prevented from accessing the student's score again. In this case, the insider still infers the correct value of the student's grade based on the old value of the student's score. We say in this case that the old value of the student's score in the insider's knowledgebase has not expired although it has been updated. However, if the Score value is changed to 91, which will change the Grade to A, the old value of Score will expire since the inference based on it is incorrect. The concept "*Expired data item*" is defined as follows.

Definition 25 (Expired Data Items). *Given the data items A and B in a relational database DB and the dependency A → B, A is called an expired data item if its value is updated to a new value such that the inferred information about B based on A's old value is incorrect.*

Checking the lifetimes of data items has a great impact on insider threat prevention and on the performance of systems. For example, suppose that a security protocol denied the request of an insider to access a data item (X) because s/he may combine it with a data item R in his/her knowledgebase to infer some unauthorized information. However, if the value of data item R has expired, the system unnecessarily denied the access to X since providing the value of X would not create a problem; rather by denying access to X, stops the user from performing his/her job on a timely basis. Similarly, ignoring the knowledgebase and granting access irrespective of the history of previous accesses and data item's lifetime may pose a threat. Thus, both these issues

should be considered when an insider requests accesses to data items.

The work by Farkas et. al [Farkas01] attempted to increase the availability of data items by checking the updates history. In their work, each insider has a history file that stores all data items, which the insider either has previously received or can disclose from the received data items. When an insider launches a query, all data items that can be received from this query are stored in the file. The data items that a user can infer are discovered by considering the current request, the history file, and the dependencies among data items. Based on the inferred data items, the system decides whether to grant or deny the requested data items. However, some data items that were accessed in the past may have been updated by others as explained before. Therefore, the inferred data items based on those expired data items would be incorrect. We should mention here that the researchers in [Farkas01] consider that a knowledge unit is expired if it is updated after the last access to it by the user. However, this dissertation states that updating the value of a knowledge unit does not always mean that its lifetime is expired. Actually, its lifetime is not expired as long as its old value can still make correct inference. Hence, their assumption may lead to the disclosure of sensitive data and failure to detect and prevent insider threat.

4.5.2 The Proposed Approach

Constructing the knowledge graph of an insider, which shows his/her knowledgebase, helps in predicting and preventing insider threat (disclosure of unauthorized information). This dissertation introduces the *Threat Prediction Graph (TPG)*, which is built based on the knowledge graph, to predict and prevent this type of insider threat. Before defining the TPG

formally, let us introduce the *Threat Prediction Value (TPV)*. A TPV is a value stored in each attribute that belongs to the TPG of the insider, and it is used to predict insider threat. A TPG is computed as follows.

$$TPV(k) = F(k) / T(k) \qquad (3)$$

where k is an attribute, F(k) is the amount of information the insider has about k, and T(k) is the threshold value of k according to the insider. T(k) represents the amount of information that the insider is allowed to get about k. TPG uses TPV to detect and prevent insider threat. An attribute is considered a threat if its TPV is greater than 1, which means that the insider can get more information than allowed about the attribute. TPG is defined as follows.

Definition 26 (TPG). *The Threat Prediction Graph (V, E, L) is a graph that is used to predict and prevent insider threat, where:*

1. *V indicates nodes such that:*
   - *The insider node represents the corresponding insider.*
   - *The second level of nodes (with labels T_i inside) represents the tables about which the insider has knowledge (tables' nodes).*
   - *The third level of nodes (with labels a_i inside) is the attributes about which the insider has knowledge (attributes' nodes).*
2. *E indicates the edges such that:*
   - *Dashed edges (arrows) represent the paths the insider follows to get knowledge about destination data items (tables or attributes).*

- *Solid edges (arrows) point to destination objects (tables or insider nodes) to which source data items (tables or attributes) belong.*

3. *L represents the TPV values of attributes' nodes.*

Figure 4.10 shows an instance of a TPG. The following points should be taken into account when analyzing a TPG:

a) A solid arrow from a table node to the insider node ($e(T_i, Insider)$) indicates that the insider has information about the table.

b) A solid arrow from an attribute node to a table node ($e(a_i, T_i)$) indicates that the attribute belongs to the table.

c) A dashed arrow from the insider node to a table node ($e(Insider, T_i)$) or from a table node to an attribute node ($e(T_i, a_i)$) indicates that the insider has direct read access to those data items.

d) A dashed arrow from an attribute node to another attribute node ($e(a_i, a_j)$) indicates that the first attribute (a_i) is used to infer information about the second one (a_j).

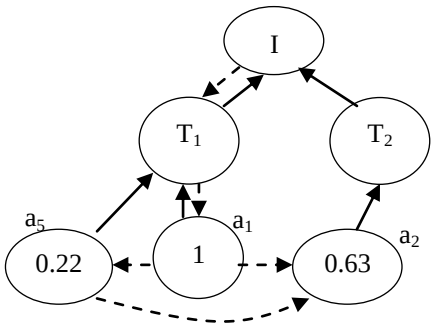

Figure 4.11. An Instance of TPG

For instance, in Figure 4.11, the insider has information about the tables T_1 (by direct access) and T_2 (by inference). Similarly, s/he can get information about the attribute a_1 by direct access,

however, s/he infers information about a_5 and a_2 using a_1, where a_2 belong to T_2 and the other two attributes belong to T_1. The values inside the attributes nodes represent the TPV for those attributes. To construct a TPG, NDIG, the KG and the set of threshold values according to the underlying insider are used. The threshold value of an attribute according to an insider represents the percentage amount of information that the insider is allowed to get about the data item, where 100% indicates that the insider can get full information about the corresponding data item, and a value less than 100% indicates that the insider can get partial information. As discussed earlier, the amount of information that an insider gets about a data item is retrieved using NDIG.

4.5.2.1 The Algorithm for Insider Threat Prediction and Prevention

Algorithm 2 shows how to detect and prevent insider threat using TPG. The algorithm uses NDIG and KG to build the TPG. Moreover, it uses the threshold values for data items according to the corresponding insider to compute TPVs.

Algorithm 4.2. Insider Threat Prediction and Prevention

Input: An insider I, the set of threshold values according to the insider, NDIG, the knowledge graph KG of the insider.

Output: The Threat Prediction Graph TPG of the insider I.

1. Initialize the set of pairs T ={(KU,TKU)}, where TKU is the threshold value about a knowledge unit KU according to the insider I, an empty set S={}
2. Recall the KG of the insider and the NDIG, initialize the TPG as TPG = KG, but without labels
3. **For** each KU ∈ V // *knowledge unit*
4. TPV(KU) = F(KU) / T(KU) //*compute the TPV of KU*
5. KU.TPV = TPV(KU) //*store the TPV inside the node*
6. **Endfor**
7. **For** each requested knowledge unit RKU by the insider

8. **If** TPV(RKU) > 1 //*threat predicted*
9. Deny this request
10. **Else** //*add RKU temporarily for further inspection*
11. V = V ∪ {T_k}, where RKU ∈ T_k // *add table T_k, where $T_k \notin TPG$*
12. E = E ∪ {e(I, T_k)} //*add an edge if e(I, T_k) ∉ E*
13. V = V ∪ {RKU} // *add a node for RKU*
14. E = E ∪ {T_k, RKU} //*add an edge to the TPG*
15. RKU.TPV = TPV(RKU) // *Store the TPV(RKU) inside its node*
16. **For** each knowledge unit KU_x that has a dependency with RKU //*add inferred attributes*
17. **If** TPV(KU_x) > 1 // *threat predicted*
18. Deny RKU and remove it from TPG
19. **Else** // *no threat so far, still needs further inspection*
20. **If** $KU_x \notin V$ // *not in the TPG*
21. **If** KU_x and RKU are not in the same table
22. Repeat steps 11 to 15 for KU_x //*add inferred attributes*
23. Add KU_x to the set S // *for further inspection (step 37)*
24. **Else**
25. V = V ∪ {KU_x} // *Add a node for KU_x*
26. E=E ∪ {e(RKU, KU_x)} // *add an edge*
27. KU_x.TPV = TPV(KU_x) // *Store the TPV(KU_x) inside its node*
28. Add KU_x to the set S // *for further inspection step37*
29. **Endif**
30. **Else** // *$KU_x \in V$, already in the TPG*
31. Add KU_x to the set S // *for further inspection step37*
32. E=E ∪ {e(RKU, KU_x)} // *add an edge*
33. Update the TPV of KU_x // *re-calculate its TPV*
34. **Endif**
35. **Endif**

36. **Endfor**

37. **For** each KU in S

38. **If** TPV(KU) > 1 // *threat predicted*

Two solutions: // *threat prevention*

39. **First**: Allow RKU but revoke insiders' accesses to a knowledge unit(s), say KU_z, that has the following properties:

(a). KU_z already exists in the knowledgebase of the insider.

(b). KU_z can be used in conjunction with RKU to compromise the unauthorized information about KU.

(c). The lifetime of KU_z is expired.

(d**)** Revoking access to KU_z preserves the security of all attributes.

OR: //*If the first solution is not possible*

40. **Second**: Deny the insider's request to RKU and recover the TPG as it was before step 7.

41. **Endif**

42. **Endfor**

43. **Endif**

44.**Endfor**

The algorithm works as follows. First, it initializes the TPG to the KG of the corresponding insider (step 2). Next, it computes the TPV for each attribute in TPG and stores the value in the attribute's node (steps 3-6). When an insider requests an access to a data item, say RKU, the algorithm checks whether the TPV of RKU is greater than 1. If TPV(RKU) > 1, the request is denied (steps 8-9). Similarly, the request is denied if RKU can be used alone to infer unauthorized information (steps 17-18). Otherwise, RKU and all data items dependent on RKU are added temporarily to the TPG (19-34). These data items are inspected further by the algorithm later in the following steps.

Before going further, we should mention here that the TPV of all data items in the TPG are recalculated after adding RKU. This is because RKU and some data items in the knowledgebase (in the TPG) of the insider may be combined to get more information about other data items. Thus, the algorithm checks if RKU can be combined with a data item(s) in the insider's knowledgebase to make the TPV of other data item(s), say KU_j , greater than 1, which indicates a threat. To prevent this threat, the algorithm introduces two solutions to solve this problem. The first solution is to deny the insider's request (RKU). The second one is to grant the insider an access to RKU, but to revoke access(es) to a data item(s), say KU_z, that can be combined with RKU to pose the threat (unauthorized information about KU_j). KU_z should have the following properties. First, it already exists in the knowledgebase of the insider. Second, it can be used in conjunction with RKU to compromise unauthorized information about KU_j. Third, its life time is expired. Finally, revoking access to it preserves the security of all attributes. Both solutions prevent insider threat; however, the second solution preserves the availability of the data items needed to execute the insiders' tasks. Thus, the second solution should be considered first, if it is possible. Otherwise, the second solution is used.

The second solution states that when a data item (KU_z) is expired, the inference based on it is incorrect. In this case, if the insider uses the old (expired) value of KU_z, which exists in his knowledgebase, and combines it with RKU to infer unauthorized information, his/her inference would be incorrect. Moreover, if the insider tries to re-read the new value of KU_z to use it in inference, his/her attempt is denied since s/he does not have permission anymore to access KU_z.

4.5.2.2 The Proof of Correctness of Algorithm 4.2

The following lemmas and theorem prove that the algorithm predicts and prevents insider threat.

Lemma 1. *Consider RKU, T(RKU) and F(RKU) as stated in the algorithm, then:*

$$\forall RKU: (T(RKU) < F(RKU)) \rightarrow Deny(RKU)$$

where Deny (RKU) means that access request for RKU is not granted.

Proof:

The proof of this lemma is fairly straight forward. Obviously, steps 8-9 states that a requested attribute is not granted if the insider can get more information than allowed about it (greater than the threshold value).

Lemma 2. *Consider KU, TPG, KG and insider I as stated in the algorithm, then:*

$$KU \in KG(I) \rightarrow KU \in TPG(I).$$

Proof:

The lemma states that every attribute about which the insider has knowledge is added to the TPG. This is obvious in step 2, which states that the TPG is initialized to the knowledge graph of the insider. Thus, since the knowledge graph contains all attributes about which the insider has knowledge, as proved in Theorem 1, the initialized TPG contains all those attributes. □

Theorem 3. *Consider RKU and TPV as stated in the algorithm, then:*

$$\forall RKU: Grant(RKU) \rightarrow \forall t \in TPG: TPV(t) \leq 1,$$

where Grant(RKU) means that RKU is granted.

The theorem states that a requested attribute is granted if the granting preserves the safety of all attributes.

Proof:

Initially, assume that no threat exists. That is, all attributes are safe. Let x represent the requested attribute. By lemma 1, if TPV(x) > 1, the request is denied, which prevents the threat and keeps all attributes secure. Otherwise, x is granted if for all attributes t in TPG, TPV(t) ≤ 1, which is proved as follows. First, x is added to the TPG by steps 11 to 15. Then, the algorithm checks what new knowledge the insider can infer using x. This is performed by investigating dependencies between x and all other attributes that are already in the knowledgebase of the insider. This is easy to accomplish since these attributes are added to the TPG (by lemma 2). Next, the algorithm updates the TPVs of the attributes about which the insider may get new (or more) knowledge. This is performed by steps 20-33. Then, steps 37-38 check whether any of the TPVs is greater than 1 (threat). If a threat is discovered, steps 39-40 suggest two solutions. First, deny the insider's request to access x and recover the state of the TPG to step 7, which is a safe state as assumed earlier. Second, allow the insider to access x, but revoke his/her access to other attribute(s), such that revoking the access moves the TPG to a new safe state. This proves that accessing a data item is allowed if granting it to the corresponding insider preserves the security of all data items in the TPG. □

4.5.2.3 An Example Scenario

Suppose that Figure 4.12 represents the NDIG of a relational database, where the table T_1 contains the attributes $\{a_1, a_5\}$, the table T_2 contains $\{a_2\}$, and the table T_3 contains $\{a_3, a_4, a_6\}$. Also, assume that the set of attributes to which the corresponding insider has direct access is $\{a_1\}$. Obviously, the knowledgebase of the insider is $\{(a_1, 100\%), (a_5, 11\%), (a_2, 5\%)\}$. The percentages of values represent the amount of information that the insider has about data items; 100% indicates exact knowledge and less than 100% means partial knowledge. Notice that the amount of knowledge about a_2 is acquired by cyclic inference. We should mention here that the weights on edges in the graph are the amount of information the insider gets if s/he has exact knowledge about source data items (left side of a dependency). Now, assume that $T = \{(a_1, 100\%), (a_2, 19\%), (a_3, 100\%), (a_4, 100\%), (a_5, 50\%), (a_6, 65\%)\}$, where T is the set of threshold values for data items according to the given insider. These values indicate the maximum amount of information that the corresponding insider is allowed to get about each data item. The initialized TPG for this insider is shown in Figure 4.13 (a).

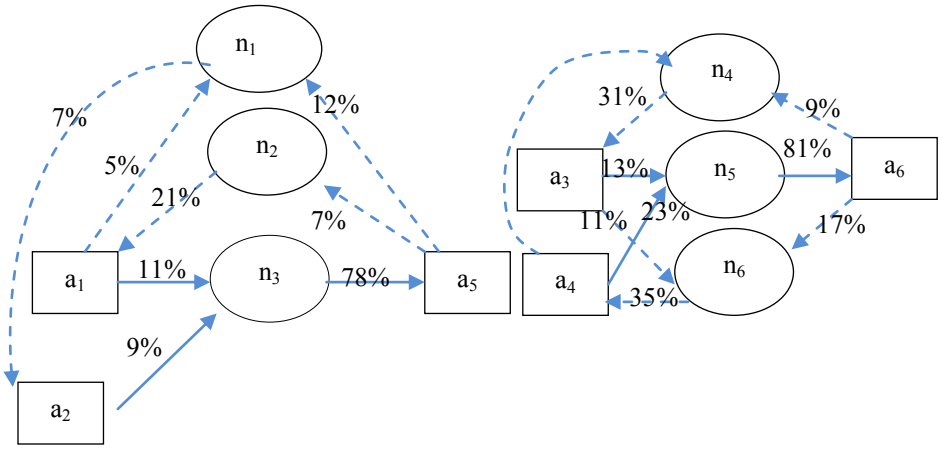

Figure 4.12. An NDIG of a Relational Database

Now, assume that the insider has requested access to a_3. Obviously, granting an access to a_3 does not form any threat as shown in Figure 4.13 (b) since the TPV will be less than or equal to 1 for all attributes when granting a_3. Bold dashed arrows demonstrate how the graph would look if the requested attribute (a_3 in this case) is granted. Notice that the TPV of a_4 is 0.35 if the insider has exact knowledge about a_6 and a_4. But since the insider has partial knowledge about a_6, the assumed value of the TPV is 0.15. Next, suppose that the insider has requested access to a_4. Obviously, the TPV of a_4 is 1, which is legal. However, granting it makes the TPG as shown in Figure 4.13 (c). In this TPG, the TPV of a_6 is greater than 1, which indicates a threat. In this case, the system has two choices. First, the system grants the insider an access to

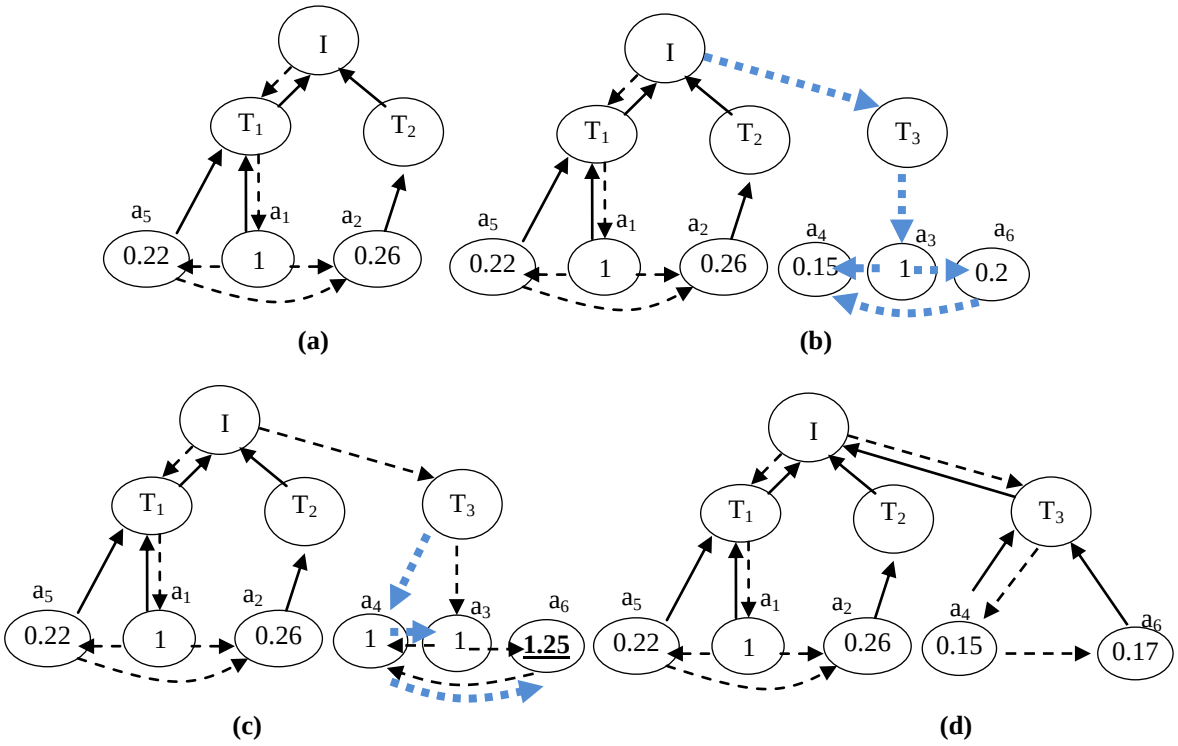

Figure 4.13. Predicting and Preventing Insider Threat Using the TPG

a_4, but revokes his/her access to a_3 (if its lifetime is expired). Second, the system denies the insider's request to read a_4. If the system chooses the first choice, the TPG of the insider will look as shown in Figure 4.13 (d). This option allows the insider to perform his/her task without limiting the availability of data items or revealing sensitive information.

4.6 Simulation

The simulation was performed using MS C#.net and SQL Server. A sample relational database of 10 tables was created manually. The dependencies and the NDIG of the database were created randomly. Similarly, the threshold values of insiders about data items were created randomly as well. The simulation was performed by choosing the number of insiders, the number of transactions, and the number of attributes in transactions at each round. The timestamps of reading or writing data items was stored to show whether data items are expired in the knowledgebases of insiders. Moreover, the amount of information that insiders get about each data item was computed and stored using the NDIG of the database. The approach was tested according to different parameters to show its effectiveness. The parameters used were the number of insiders in the system, the number of transactions, and the percentage of write operations in transactions. For the same values of parameters, the simulation was executed 100 times and the average was taken as the result. We should mention here that all threats were prevented either by finding and removing expired data items from knowledgebases or by denying read accesses. However, the percentage of prevented threats shown in the figures below indicates the number of threats that was prevented by finding and removing expired data items

over the total number of threat, which shows the effectiveness of using the proposed approach in preventing threat without limiting the availability of data items.

Figure 4.14 shows the results of the simulation with different number of insiders. The number of transactions is fixed at 250 at each round. The results show that when the number of transactions and the number of insiders are fixed, the performance of the proposed approach improves as the percentage of write operations increases. This is due to the fact that when the number of write operations increases, the number of expired data items increases as well. Thus, the probability of finding an expired data item to prevent a threat using the proposed approach gets higher. Contrarily, the figure shows that there is no trend when the number of insiders increases. The analysis of this result is as follows. When insiders execute a small number of transactions, the data items in their knowledgebases will be few. Fewer data items in an insider's knowledgebase leads to two conclusions. First, it means a smaller number of threats is possible by the insider. That is, the probability of using data items in knowledgebase to pose a security violation gets smaller. Second, it means that there is less probability of finding an expired data item when a threat arises since few updates are executed when the number of transactions is small. These two opposite effects keep the percentage of prevented threats almost stable in general as the number of insiders increases or decreases.

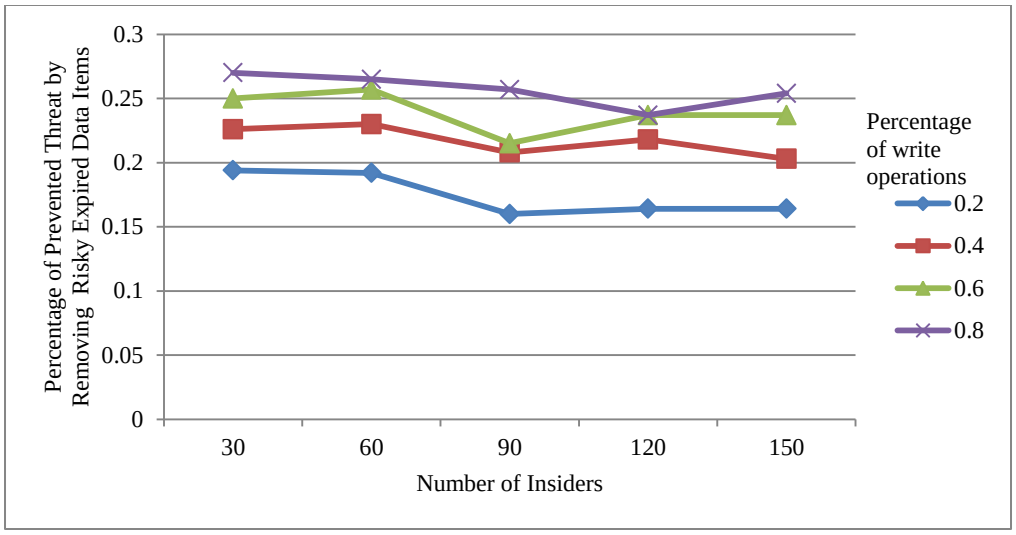

Figure 4.14. The Percentage of the Prevented Threat by Removing Expired Data Items according to Different Percentages of Write Operations and Different Number of Insiders

Figure 4.15 shows the results of the simulation according to different number of transactions and different percentage of write operations, whereas the number of insiders is fixed at 20. The figure shows that for the same number of transactions and insiders, the percentage of prevented threat by removing risky expired data items increases as the percentage of write operations increases. The analysis of this result is similar to that of Figure 4.14. In addition, Figure 4.15 shows that the number of prevented threats increases as the number of transactions increases. At first glance, this result seems strange since an increase in the number of transactions causes both the number of write and read operations to increase. Thus, data items are expired and refreshed quicker when executing the transactions. That is, no general trend of the prevented threats should be detected. However, this assumption is incorrect as shown in the figure and the reason is as follows. Since there are 20 insiders in the system, when an insider refreshes an expired data item in his/her knowledgebase by executing read operations, there are 19 other insiders available to update the expired data item and make it expire. Thus, the probability of preventing threat by finding and

removing expired data items increases. In summary, increasing the number of transactions increment the probability of re-reading a data item by an insider, but it greatly increases the probability of updating and expiring the data item by other insiders.

The simulation shows that the proposed approach prevents all detected insider threats. Moreover, it shows the effectiveness of the proposed approach in preventing insider threats without limiting the availability of data items (without denying read access requests). As shown in the figures, the percentage of the prevented threats ranges from 8% to 30% depending on the number of transactions and the percentage of write operations in transactions.

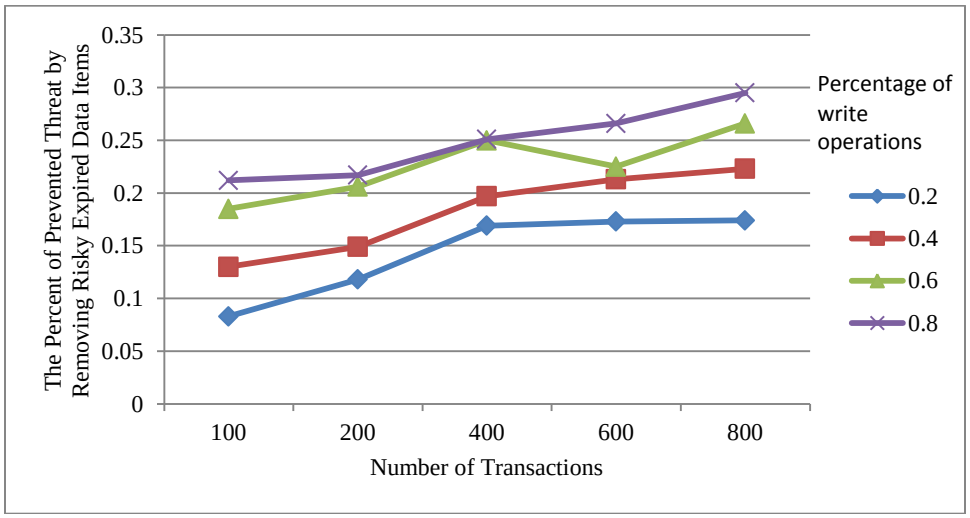

Figure 4.15. The Percentage of the Prevented Threat using the Proposed Approach under Different Percentages of Write Operations and Different Number of Transactions

5. INSIDER THREAT: UNAUTHORIZED MODIFICATIONS ATTACKS

5.1 Introduction

By discovering constraints on dependencies, an insider may be able to modify a dependent data item, to which s/he has no write access, to a desired value by updating the pre-cursor data item(s). For instance, consider the dependency {Rank, Experience} → Total_Salary as discussed earlier. Assume that an insider has a write access on Rank and Experience, but s/he has no write access on Total_Salary. In addition, assume that the insider is familiar with the dependency and constraints. In this case, the insider can modify the value of Total_Salary for an academic staff to the value s/he prefers, which can be performed by choosing the appropriate values of Rank and Experience for the academic staff. This section discusses this problem and suggests possible solutions.

5.2 Insiders' Modification-Lists

A modification-list determines which data items an insider can modify. It is constructed based on the different levels of granularity of relational databases. Based on the concepts of Hot and Safe clusters, the followings are concluded. First, if an insider is granted write access to a table that belongs to a Hot Cluster, s/he can make changes in all other tables in that cluster. Second, if an insider is given a write access to a table in a Safe Cluster, s/he cannot modify any other table (without having direct write access to it) that belongs to the same cluster. Finally, if an insider gets write access to a table in some cluster, s/he still can modify other dependent tables that belong to different clusters. This case occurs when the dependency between tables is a One_Way

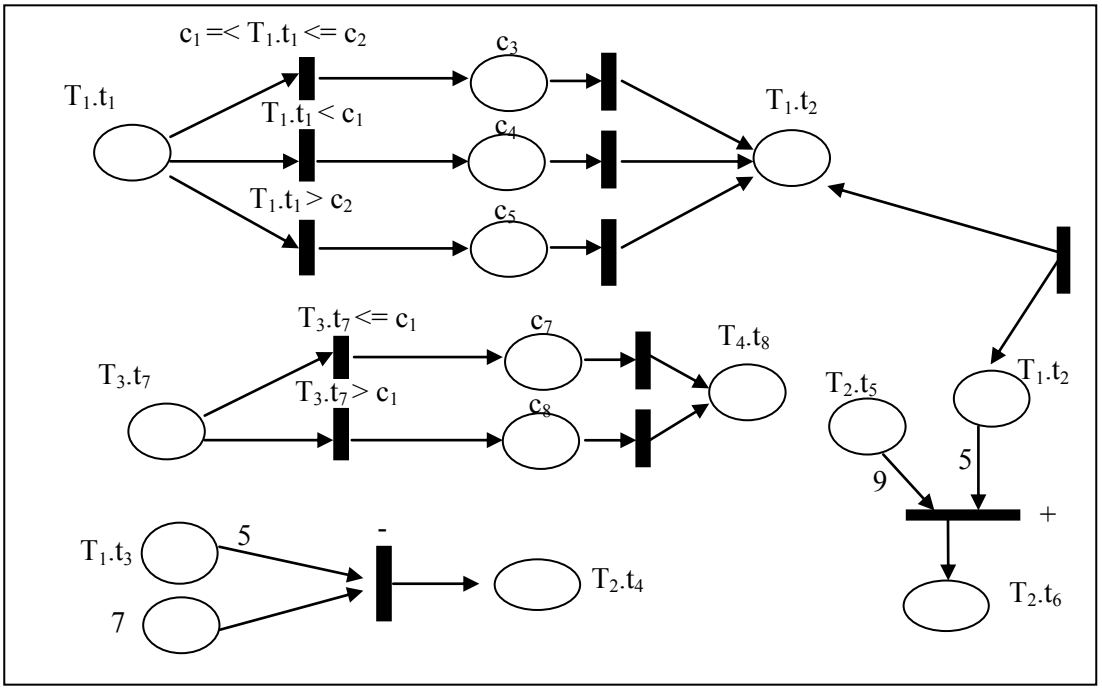

Figure 5.1. A Constraint and Dependency Graph CDG

dependency. In addition to the above conclusions, an insider can make changes in other tables transitively. The modification-list of an insider does not necessarily mean that the insider can make arbitrary changes to the associated tables. To know what changes are possible, dependencies among attributes in those tables and associated constraints should be investigated. To do this, the dependency graph CDG is used. For instance, suppose that an insider has full write access on table T_1 as shown in Figure 5.1. Both tables T_1 and T_2 are added to the modification-list of the insider since they have a dependency relationship. The insider can change data in T_1 as and when s/he wishes, whereas his/her write access to T_2 is limited by the dependency between the two tables. To determine what changes the insider can make in T_2, dependencies between attributes in both tables should be investigated. Clearly, s/he can modify attributes t_4 and t_6 only in T_2.

A similar scenario is used for items at the records level. If the insider has write access to a record, then this record must be in his/her modification-list. In addition, other records that depend on this record must be there as well. Nonetheless, this does not mean that the insider can change data in all fields in those records. To investigate what information the insider can change, dependencies among attributes should be investigated.

5.3 The Modification Algorithm

Algorithm 5.1 shows how to construct *Modification Graphs*, which represent modification-lists at different levels of granularity. The algorithm uses the CDG and Dependency Matrix to construct the modification graphs of insiders. In addition, it uses Hot and Safe clusters to facilitate the construction process. A Modification Graph is defined formally as follows.

Definition 27 (MG). *The Modification Graph MG (V, E) is a graph that demonstrates the data items (tables and attributes) that an insider can modify directly or indirectly in a relational database system, where:*

6. *V indicates nodes such that:*
    - *The insider node represents the corresponding insider.*
    - *The second level of nodes (labeled T_i) represents the tables which the insider can modify (Tables' nodes).*
    - *The third level of nodes (labeled a_i) shows the attributes which the insider can change (attributes' nodes).*
7. *E indicates the edges such that:*

- *Dashed edges (arrows) represent the paths the insider follows to modify destination data items (tables or attributes).*
- *Solid edges (arrows) point to destination objects (tables or insider nodes) to which source data items (attributes) belong or which the insider can change.*

The modification algorithm starts by adding the insider as the root of the modification graph. Next, it adds the tables which the insider can change directly or indirectly (using dependencies), at the second level. For each table at the second level, the algorithm determines to which attributes the insider has a write access (direct or indirect) and inserts them at the third level.

Consider the CDG as shown in Figure 5.1 and assume that the insider has a write access to T_1. Figure 5.2 shows the modification graph of the insider. Dashed arrows represent the paths that the insider follows to make changes. For instance, the insider can modify attribute t_2 directly, whereas s/he can modify t_6 indirectly through t_2.

Algorithm 5.1. The Modification Algorithm

Input: An insider I, Dependency Matrix, CDG, Hot and Safe clusters, Set of tables to which the insider has write access S.
Output: The Modification graph MG of the insider *I.*

1. Initialize the MG = (V,E) , where V={I} and E={}.

2. **For** each table T_k in S //*add direct write accessed tables to the graph*

3. $V = V \cup \{ T_k \}$ //*add the node T_k to MG*

4. $E = E \cup \{e(I, T_k)\}$ // add edge $e(I, T_k)$ to the MG

5. **For** each attribute $t \in T_k$ that the insider has a write access to it // *add directly accessed attributes to the MG*

6. $V=V \cup \{ t \}$ //add the node *t* to MG

7. E=E ∪ {e(T_k, t)}// *add edge e(T_k, t) to the MG*

8. Endfor

9. Endfor

10.For each T_k in S **do** // *tables in S*

11. For each Safe Cluster R to which T_k belongs

12. Exclude all tables in R (that does not belong to S) from the MG of the insider

13. Endfor

14. For each Hot Cluster H to which T_k belongs

15. For each table $T_m \in H$

16. V = V ∪ { T_m } //*add the node T_m to the MG*

17. E = E ∪ {e(I, T_m)} // *add the edge e(I, T_m) to the MG*

18. Endfor

19. For each attribute t ∈ T_m that the insider can change depending on direct dependencies//*add t to the MG*

20. V= V ∪ { t } //*add the node t to MG*

21. E= E ∪ {e(T_m, t)} //*add the edge e(T_m, t) to the MG*

22. E= E ∪ {e(d, t)}, where d is an attribute that belongs to T_k and on which t depends

23. Endfor

24. Endfor

25. For each other table T_s that depends (one-way) on T_k//*add tables from other clusters*

26. Repeat steps 15 to 23 for the table T_s

27. Endfor

28. For each table T_j that depends transitively on T_k (transitive dependency) // *transitive change*

29. V=V ∪ {T_j} //*add the node T_j to MG*

30. E=E ∪ {e(I, T_j)}// *add edge e(I, T_j) to the MG*

31. Endfor

32. For each t ∈ attributes(T_j) that the insider can change it transitively (transitive dependency) //*add t to the MG*

33. V = V ∪ {t} //add the node *t* to MG

34. E = E ∪ {e(T_j, t)}//add edge e(T_j, *t*) to the MG

35. E = E ∪ {e(f, t)}, where f is an attribute that belongs to a table in MG and on which t depends directly

36. Endfor

37.Endfor

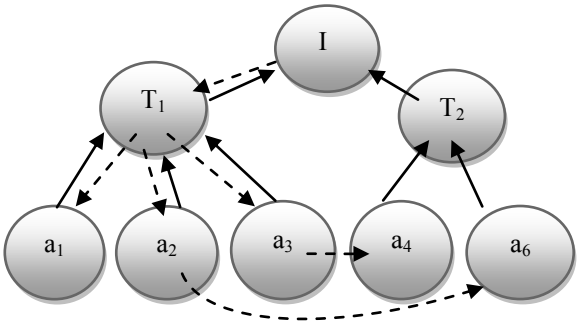

Figure 5.2 A Modification Graph of an Insider.

5.4 The Proof of Correctness of Algorithm 5.1

Insiders can modify data items in different ways. First, they can modify data items using their privileges. Second, they can modify data items indirectly using direct dependencies. Finally, they can modify data items transitively using transitive dependencies. The following theorems prove that Algorithm 5.1 considers all these ways when building modification graphs. We should mention here that the proofs are similar to the proofs in Algorithm 4.1. However, Algorithm 5.1 deals with modification graphs instead of knowledge graphs.

Theorem 1. *Given a table B in a relational database DB, and the modification graph MG of an insider, then:*

$$B \in MG \Leftrightarrow B \in D(I) \vee B \in DD(A) \vee B \in TD(A)$$

where D(I) is the set of tables to which the insider has direct write access, the table A ∈ D(I), and DD(A) and TD(A) are the set of tables that depend directly and transitively respectively on A.

The theorem lists the three ways which insiders follow to modify data items. The following proof verifies that the algorithm adds all tables which the insider may modify using those ways.

Proof:

(→) Suppose that A ∈ D (I) and DB has the following dependencies:

1) A→B, which means that B depends directly on A.

2) A⇢ C, which means that C depends transitively on A.

It is obvious that A∈ MG by steps 2 to 4, which add every directly accessed table (write access) to the modification graph. This proves the part of directly accessed tables with a write privilege. In the case of table B, let H(A) be a Hot Cluster that contains A. Now, since A → B, then either B∈ H(A) (A and B have cyclic dependency), or B ∉ H(A). In the first case, B ∈ MG by steps 14 to 18. In the second case, B ∈ MG by steps 25 to 27. This proves the part of directly dependent tables. In the case of C, where C depends transitively on A, C ∈ MG by steps 28 to 31. This proves the part of transitively dependent tables. In summary, all tables which the insider can modify are added to the modification graph. □

(←) (Proof by Contradiction) This part proves that there is no table added to the modification graph of the insider but those added in the previous part. Suppose that ∃B ∈ MG: (B ∉ D(I) ∧ B ∉ DD(A) ∧ B ∉ TD(A)),where A ∈ D(I). In this case, B should exist in a Safe Cluster (Definition 8). Thus, B ∉ MG by steps 11 to 13, contradiction.□

As discussed earlier, if a table exists in the modification graph of an insider, this does not mean that the insider can modify every attribute in that table. The following theorem clarifies this claim.

Theorem 2. *Given an attribute k, where $k \in B$ for some table B in a relational database DB, and the modification graph MG of an insider I, we have,*

$$k \in MG \Leftrightarrow k \in DA(I) \vee k \in DDA(s) \vee k \in TDA(s)$$

where DA(I) is the set of attributes to which the insider has direct write access, the attribute $s \in DA(I)$, DDA(s) and TDA(s) are the set of attributes that depend directly and transitively on s respectively.

Proof:

(→) Suppose that DB has the following dependencies:

1) A→B ... → T, where A, B and T are tables in DB, and A ∈ D(I).
2) k→ r ... → z, where k, r and z are attributes, and k ∈ A, k ∈ DA(I), r ∈ B and z ∈ T.

First, steps 5 to 8 state that ∀ k: (k ∈ DA(I) : (k ∈ A ∧ A ∈ D(I))) → k ∈ V(MG) ∧ e(A, k) ∈ E(MG). This proves the part of directly accessed attributes with a write privilege. Second, steps 19 to 23 state that ∀ r: (r ∈ DDA(k) : (k ∈ A ∧ A ∈ D(I) ∧ r ∈ B ∧ B ∈ DD(A))) → r ∈ V(MG) ∧ {e(B, r), e(k, r)} ∈ E(MG). This proves the part of directly dependent attributes. Finally, steps 32 to 36 state that ∀ z: (z ∈ TDA(k) : (k ∈ A ∧ A ∈ D(I) ∧ z ∈ T ∧ T ∈ TD(A))) → z ∈ V(MG)) ∧ {e(T, z), e(k, z)} ∈ E(MG). This proves the third case. □

(←) (Proof by Contradiction) Suppose that ∃k ∈ MG: (k ∉ DA(I) ∧ k ∉ DDA(s) ∧ k ∉ TDA(s)), where s ∈ DA (I). In this case, there are four cases:

1) k ∈ A , where A ∈ D(I) but k ∉ DA(I). In this case, k is excluded (not added) using steps 5 to 8.
2) k ∈ B , where B ∈ DD(A) but k ∉ DDA(s) for some A ∈ D(I) and s ∈ DA (I). In this case, k is excluded using steps 19 to 23.
3) k ∈ T , where T ∈ TD(A) but k ∉ TDA(s) for some A ∈ D(I) and s ∈ DA(I). In this case, k is excluded using steps 32 to 36.
4) k ∈ A and A ∈ S(P), where S(P) is a Safe Cluster of a table P that the insider can modify. In this case, k is excluded using steps 11 to 13.

Obviously, all mentioned cases contradict the assumption.□

5.5 Preventing Malicious Modifications

Preventing malicious modifications can be handled in two ways. The first method is to hide the dependencies that may be used by insiders to launch unauthorized modifications. The second method is not to grant insiders write accesses to data items that may be used to make unauthorized modifications to sensitive data items.

5.5.1 Hiding Dependencies

As discussed earlier, the discovery of dependencies by insiders may pose a threat; it allows them to make changes to unauthorized data items. For instance, the insider who has access to the Rank attribute can change the Salary attribute of an academic staff. However, if the insider is not familiar with the dependency, s/he may not make unauthorized modifications. Actually, an

insider can still make changes but these changes will be random, and a random change will generate suspicion. Determining which dependencies should be hidden depends on the sensitivity of the data items. That is, some data items are not important enough for insiders to be interested in changing. The level of importance defines the sensitivity of data items. A *Sensitive Data Item* is defined as follows.

Definition 28 (Sensitive Data Item). *A Sensitive Data Item is a data item which insiders may be interested in changing due the importance and secrecy of the information that it contains.*

Determination of the sensitivity of a data item is performed by administrators who can assign values between 0% for insensitive data items and 100% for highly sensitive data items. Administrators should consider the importance of data items when assigning sensitivity values. To determine which dependencies should be hidden from an insider, the *Sensitivity and Dependency Graph (SDG)* is introduced, which shows the dependencies among attributes in different tables without revealing any further details about them, such as constraints. But purposely, it contains the sensitivity values of different attributes. This facilitates determining the *Cut*, which is defined as follows.

Definition 29 (Cut). *Given a set of dependencies S in a relational database, a Cut is a set of dependencies* $C \subseteq S$ *that should be hidden from the insider under consideration.*

Those edges (dependencies) have destination attributes with sensitivity values greater than a predefined threshold for the insider under consideration. Hence, when an insider has a write access to a *Hot Attribute* [White09b], many dependencies need to be hidden from him/her.

Figure 5.3 shows an example of determining a Cut in SDG. The weights on edges indicate the sensitivity of the destination attributes. The Cut shows that the dependencies {X→R, X→Q, P→Z} should be hidden from the insider who has a write access on X and not on Z, Q and R. In addition, the insider should be prevented from collaborating with insiders who have access to attributes Z, Q and R.

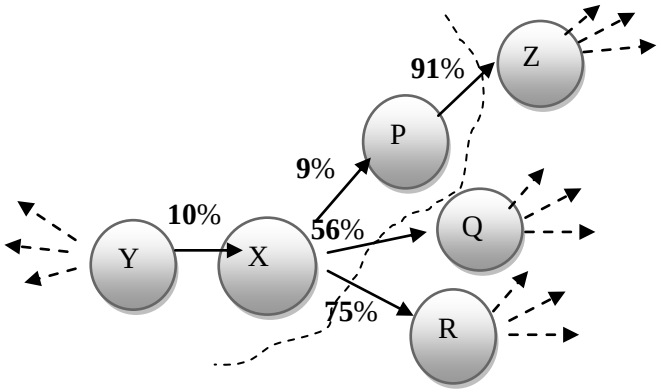

Figure 5.3. Determining a Cut in the Sensitivity and Dependency Graph.

5.5.1.1 The Algorithm

Algorithm 5.2 shows how to determine which dependencies should be hidden from an insider. To determine a Cut, a threshold value should be set first. Then, a Breadth First Search is used, which starts from the attribute on which the insider has a write access to determine which edges belong to the Cut (step 7). Strictly speaking, all edges that have sensitivity values greater than the

threshold value are added to the Cut (steps 9-10). For instance, assume that the threshold value in the example (in Figure 5.3) is 50%. Thus, all dependencies in the Cut have dependent attributes with sensitivity values greater than 50%.

Algorithm 5.2. The Cut Algorithm.

Input: CDG, Set of attributes S, an insider *I*.

Output: A Cut C.

1. Let X be the attribute to which the insider has a write access
2. Initialize the Sensitivity and Dependency Graph using the CDG
3. Initialize a cut C={}
4. Initialize S = {X}
5. **While** S ≠ {}
6. Pick an attribute Z from S
7. Run a breadth first search on Z
8. **For** each attribute Y ∈ adjacent(Z) *// the edge e(Z, Y) ∈ SDG*
9. **If** sensitivity(e(Z,Y)) > threshold
10. C = C ∪ { e(Z,Y) } *// add the edge to the SDG*
11. **Else**
12. S=S ∪ {Y}
13. **Endif**
14. **Endfor**
15. **Endwhile**
16. Return C as the cut

5.4.1.2 The Proof of Correctness of Algorithm 5.2

The following theorem proves the correctness of the algorithm.

Theorem 3. *Let X be an attribute to which an insider has a write access, U and V are two other attributes, and C is a Cut. Then:*

$e(U,V) \in C \Leftrightarrow (\exists P(X,U):(\forall Z \in P(X,U):Sensitivity(Z) < Threshold(Z)) \wedge (Sensitivity(V) > Threshold(V))$, where P(X,U) denotes a path from the attribute X to an attribute U.

This theorem states that a Cut contains an edge e(U,V) if and only if there is a path $X \rightarrow U$, such that all attributes along this path have a sensitivity value less than the threshold value and the sensitivity of V is greater than the threshold value. Hence, the threshold value of an attribute relative to the insider under consideration determines whether the attribute is sensitive or not for the insider.

Proof:

($\rightarrow$) Suppose that $e(X,Y) \in P(X,U)$. In addition, assume that Sensitivity (Y) $\leq$ Threshold(Y). In this case, $Y \in S$ by steps 11 - 12. Later, Y will be picked from S since steps 5 - 6 pick a vertex from S recursively until S is empty. Now, by steps 8 and 11 - 12, $\forall K \in$ Adjacent(Y) $\wedge$ Sensitivity (K) $\leq$ Threshold(K) $\rightarrow K \in S$. Continuously, the algorithm picks vertices on the path $X \rightarrow U$ as long as their sensitivity is less than or equal to the threshold value. However, it stops checking the adjacent vertices of any vertex, say L, if the sensitivity of L is greater than the threshold value, which is clear in steps 9-10. When reaching U, $\forall$ R: $R \in$ Adjacent(U) $\wedge$ Sensitivity (R) > Threshold (R) $\rightarrow$ (U,R) $\in$ C (by steps 9 - 10). Thus, since sensitivity(V) > Threshold(V), the edge e(U,V) is added to the Cut. This completes the first part of the proof.

(←) (Proof by Contradiction) Suppose that $e(U, V) \in C \wedge (\forall P(X,U) : (\exists Y \in P(X,U) :$ Sensitivity(Y) > Threshold(Y)). Now, let Z be the predecessor of Y. In this case, $e(Z, Y) \in C$ by steps 9 -10, and hence, Y will not be added to the set S as shown by steps 9-13. As a result, since $Y \in P(X,U)$, U will not be reached. Thus, $e(U,V) \notin C$, which is a contradiction. □

This proves the correctness of the algorithm. In summary, Theorem 3 proves that a Cut contains an edge, say e(U,V), if the following conditions are satisfied:

1) Its endpoints are reachable from the vertex, say X, to which the insider has write access.
2) There is a path from X to V, such that all of the vertices along that path (except V) have a sensitivity value less than or equal the threshold value.
3) The sensitivity value of the destination of the edge (V in this case) is greater than the threshold value.

5.5.2 Denying Write Access Requests

It may not always be possible to hide dependencies. In these cases, the solution is not to grant insiders write accesses on data items in which a change may cause a change in sensitive data items. For instance, using the graph in Figure 5.3, granting an insider write access to the data item X enables the insider to make changes in data items P, Z, Q and R. Thus, if some of these data items are sensitive, and hiding dependencies is not possible, the insider should not get write access to X.

Insiders can make approximate or exact changes to unauthorized data items based on dependencies and/or constraints they can discover. For instance, consider the dependencies {Rank →Base_Salary, (Base_Salary, Experience) → Salary}, where Salary = Base_Salary + 100 * Experience. Suppose that the Salary attribute is a sensitive data item. Assume also that the insider under consideration is familiar with the corresponding dependencies and constraints. Now, if the insider has write access to Rank only, s/he can change the corresponding Salary to an amount close enough to what s/he wishes. Whereas by having a write access to Experience, s/he can make some minor changes to Salary. On the other hand, by having a write access to both the Rank and Experience, the insider can change Salary to any value s/he wants. Administrators should take this into account when granting write access to data items. However, denying write access to some data items may affect the tasks the insiders are able to perform.

Modification Graphs MGs show how to predict an unauthorized modification threat. For example, using the CDG in Figure 5.1, suppose that an insider has write access to the attribute a_1 in table T_1. In this case, the MG of the insider is shown in Figure 5.4, which shows that the insider can change attributes a_2 and a_6 in tables T_1 and T_2 respectively although s/he may not have write access to these attributes. Thus, if one of these attributes is sensitive, administrators may deny the insider's write access to a_1 in order to avoid the threat. However, as discussed earlier, denying write access may hinder the performance of some insiders and also reduce the availability of data items. Thus, the preferable approach is to hide the dependencies, if possible, instead of denying write accesses.

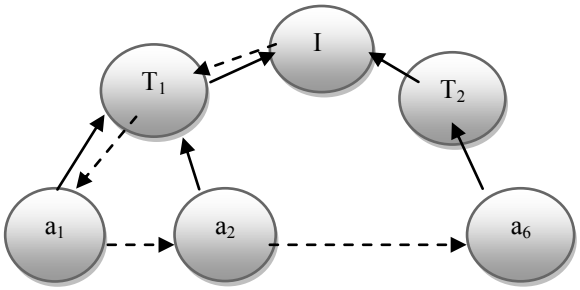

Figure 5.4 A Modification Graph.

5.6 An Example Scenario

This section introduces a simple real world example of using the Cut algorithm to prevent insider threat without limiting the availability of data items. Consider the example in Figure 5.1, suppose that the sensitivity values of the data items are as shown in Table 5.1. Sensitivity values indicate the importance and the secrecy of data items as discussed earlier. Figure 5.5 represents the SDG for the given database based on the dependencies and the sensitivity values of the data items. As discussed earlier, weights on edges represent the sensitivity of the destination data items. Notice that the starting data items (Rank and Number of dependents) do not have sensitivity values. This is because the SDG is used to show the hidden threat when granting write accesses to some data items (the starting data items). However, if a write access on those data items is requested by an insider who is not allowed to modify them, his/her request is denied without constructing the corresponding SDG.

TABLE 5.1. SENSITIVITY VALUES ACCORDING TO THE INSIDER K

Data item	**Sensitivity**
Rank	20%
Base_Salary	90%
Experience	10%
Number of Dependents	30%
HI_Premium	90%
Salary	100%
Net_Salary	100%
Tax	10%

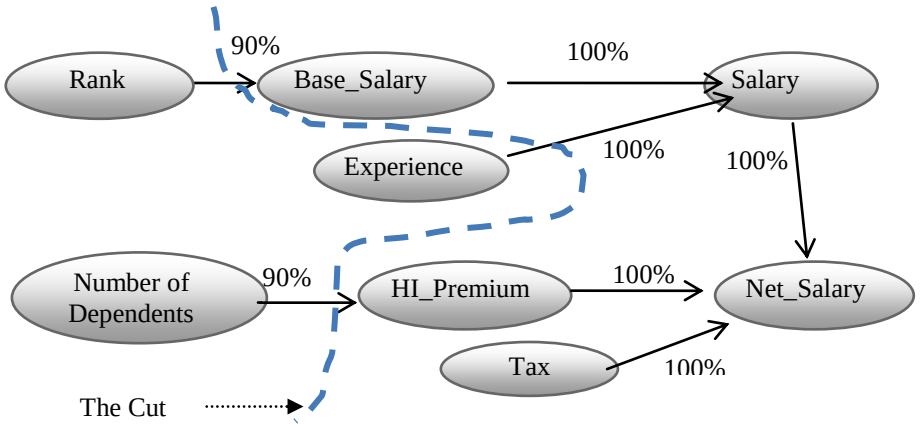

Figure 5.5.The SDG of the Academic Staff Database in Figure 5.1

Assume that the threshold value of the insider (Bob) is 50%, which means that he is not allowed to modify data items with sensitivity values greater than 50%. Now, suppose that Bob requests a write access on Rank and Experience attributes and on the Dependents table. In this case, if he is given write access to those data items, he can indirectly modify the sensitive data items: Base_Salary, Salary, HI_Premium and Net_Salary, which is a threat. It happens if Bob is familiar with the dependencies. Thus, to prevent this threat, we should ensure that some dependencies are hidden from Bob before giving him the requested write accesses. To show which dependencies should be hidden to prevent the threat, the Cut algorithm is used. Using the algorithm, the set of dependencies that should be hidden from Bob is shown in Figure 5.5 by a dashed line. By hiding those dependencies, the requested accesses can be granted and the threat is minimized or prevented. However, if it is not possible to hide those dependencies, Bob's access requests should be denied. This simple example shows the effect of hiding some dependencies to prevent insider threat. In addition, it shows how hiding dependencies increases the availability of data items so that insiders can perform their jobs without limiting their performance.

5.7 How Insiders Discover Dependencies

Insiders may discover dependencies in several ways. First, they may discover dependencies by accessing the metadata of a relational database directly, which is fairly straight forward. Second, they may discover dependencies by collaborating with other insiders in the same organization. For instance, suppose that Alice has a write access to a table T_1 and Bob has a read access on table T_2. In addition, assume that neither of the two insiders is familiar with the dependencies between the two tables. Now, suppose that Alice needs to modify some value in T_2. To do this, she can collaborate with Bob to check whether there is a dependency between the two tables; Alice makes changes in T_1 until a change happens in T_2. Then, Bob informs Alice about the change that happens in T_2. As a result of this operation, the collaborative insiders can discover the dependencies as well as the constraints between the two tables. Figure 5.6 demonstrates this process. Discussions about the prevention of collaborative attacks are beyond the scope of the dissertation.

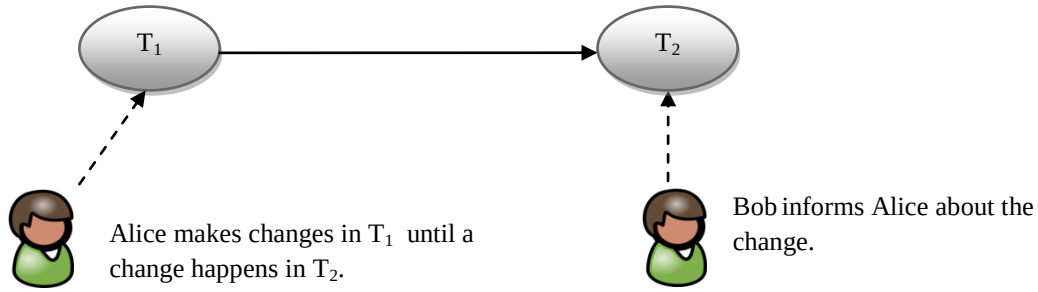

Figure 5.6. Collaborative Attacks.

Finally, an insider may discover dependencies by inferring relationships among data items that s/he retrieves. Moreover, s/he can infer the constraints on dependencies partially or totally. For instance, suppose that the relationship between the Rank of an academic staff and his/her Base_Salary is as shown in Figure 5.7. Assume that Alice has accessed tuples 1 and 3. In this

case, she may assume that there is a dependency between the Rank and the Base_Salary, and that assistant professors have a Base_Salary of 75K. Moreover, she updates her knowledge about dependencies and constraints when she accesses more tuples.

Name	Rank	Base_Salary
Jeff Mayor	Assistant Prof.	75K
Nancy Bishop	Prof.	100K
Dale Bush	Assistant. Prof.	75K
Gordon Thompson	Prof.	100K

Figure 5.7. Academic Staff's Base_Salary

5.8 Hiding Dependencies: When and How?

Hiding dependencies requires preventing the operations that may expose them. However, only sensitive dependencies need to be hidden, where a sensitive dependency is a dependency that may be used to make malicious modifications to sensitive data items. Obviously, hiding dependencies may limit the availability of data items and insiders' tasks as well. Thus, this process should be performed when the cost of allowing the discovery of a sensitive dependency is greater than the cost of hiding it. For instance, in Figure 5.5, if an insider has a write access to Rank only, having him/her discover the dependency between Rank and Base_Salary would be costly from security viewpoint. This is because modifying the Rank changes the value of Base_Salary, Salary, and Net Salary, which are unauthorized sensitive data items. To prevent the threat illustrated in this example, this dissertation proposes two solutions. The first one is to prevent the corresponding insider from discovering the dependency, which can be achieved by preventing him/her from getting a read access to the Base_Salary. In this case, the insider can get a write access to the Rank attribute only without posing a threat. The second solution is not to grant the insider a write access on the Rank attribute and to allow him/her to read the data item

(Rank, Base_Salary). However, both solutions limit the availability of data items. That is, there is a cost in terms of availability; but not in terms of security.

In light of the previous discussion, preventing the operations that may lead to exposure of sensitive dependencies or update of sensitive information has a cost on the availability of data items and also on the insiders' job performance. On the other hand, allowing these operations may help insiders in making malicious modifications to unauthorized data items, which has a cost to the security of the system. Therefore, to help in decision making process, the cost of each possible solution must be computed and the one with the least cost should be chosen. The cost of a solution can be measured according to its effect on the availability and on the security of data items. Formula 1 computes the cost of a solution.

$$Cost\ (S)=\sum_{i=1}^{n}(Imp(i)*Wa)+\sum_{j=1}^{m}(Sensitivity(j)*Ws) \qquad (1)$$

where S indicates the solution chosen, i represents the attribute that is limited (prevented) by applying S, Imp(i) indicates the importance of the data item i according to the insider under consideration, which represents the necessity of i in performing the insider's tasks. For example, the data items that are used by the insider to perform a group of tasks are more important than data items that are used to perform a single task. This value may be given either by the insider or assigned by the system based on the tasks that should be performed by the insider. The term j represents an unauthorized attribute that may be exposed by applying S, sensitivity(j) shows the sensitivity of the attribute j. W_a and W_s indicate the weights associated with the availability and sensitivity respectively. These values are used to determine which is preferable between limiting

the availability of attributes and exposing sensitive information. Hence, these values may differ according to attributes, and are assigned by the system. Likewise, the sensitivity of an attribute is assigned by estimating the damage that may be caused by revealing or modifying the attribute. A value for sensitivity is assigned by the system as well.

Name	**Rank**	**Importance /10**	**Sensitivity /10**	**Base_Salary**	**Importance /10**	**Sensitivity /10**
Jeff Mayor	Assistant Prof.	5	1	75K	2	10
Nancy Bishop	Prof.	5	1	100K	2	10
Dale Bush	Associate Prof.	5	1	80K	2	10
Gordon Bush	Prof.	5	1	100K	2	10

Figure 5.8. Academic Staff Base_Salary

To clarify the concept, consider Figure 5.8, which represents the sensitivity and the importance values of the attributes shown in Figure 5.7 for the insider (Bob). Assume that W_a and W_s are 2 and 3 respectively. Now, suppose that Bob has permissions to read all attributes of all records and modify the Rank attribute of all records, but he is not allowed to modify the Base_Salary attribute. Obviously, allowing the insider to access what he is allowed to access poses threat. Strictly speaking, Bob may infer the dependency [Rank →Base_Salary] and associated constraints. In this case, Bob may use this knowledge and his write privilege on the Rank attribute to make the changes he desires to the Base_Salary attribute without having a write access to it, as discussed earlier. Although this constitutes a threat, it may be acceptable in some situations. Thus, this is the first solution (S_1) to the problem of having the insider discover the dependency and make malicious modifications. The second solution (S_2) is to grant Bob a read access on data items Rank, and Base_Salary and revoke the write access that he has on Rank. In

this case, Bob can discover the dependency, but he is not able to make malicious modifications. The third solution (S_3) is to deny the read access that Bob has on the Base_Salary attribute and grant him a write access to Rank; this hides the dependency from him, and allows him to modify the Rank. Thus, Bob cannot discover the dependency and he is not able to make malicious modifications to Base_Salary. To choose the best solution, the cost of each one is calculated using Formula 4 as follows, where the number 4 on the summation symbol indicates the number of records in the table.

- Cost(S_1) = 0 + $\sum_1^4(10 * 3)$ = 120.
- Cost(S_2) = $\sum_1^4(5 * 2)$ + 0 = 40.
- Cost(S_3) =$\sum_1^4(2 * 2)$ + 0 = 16.

Notice that the first solution is very costly. Obviously, the best solution is S_3, which hides the dependency from Bob. That is, limiting the availability of some data items is better than allowing him to make malicious modifications according to the cost estimation.

5.9 Experiments and Results

To test the efficiency of the model, a simulation was performed using MS C#.net and SQL Server. A sample relational database of 10 tables was created manually. The dependencies and the NDIG of the database were created randomly. Similarly, the access permissions, the importance and the sensitivity of data items according to each insider were created randomly as well.

The simulation was performed by choosing the number of insiders, the number of transactions, the range of attributes per transaction and the weights of availability (W_a) and sensitivity (W_s). The model was tested according to different parameters to show its effectiveness. The parameters used are the number of insiders in the system, the number of transactions, and the percentage of write operations in transactions. For the same set of parameters' values, the simulation was executed several times and the average was taken as the result. We should mention here that all risky transactions were caught and prevented using the proposed approach (when preventing threat is less costly than allowing it). Figures 5.9 and Figure 5.10 show the results of the simulation.

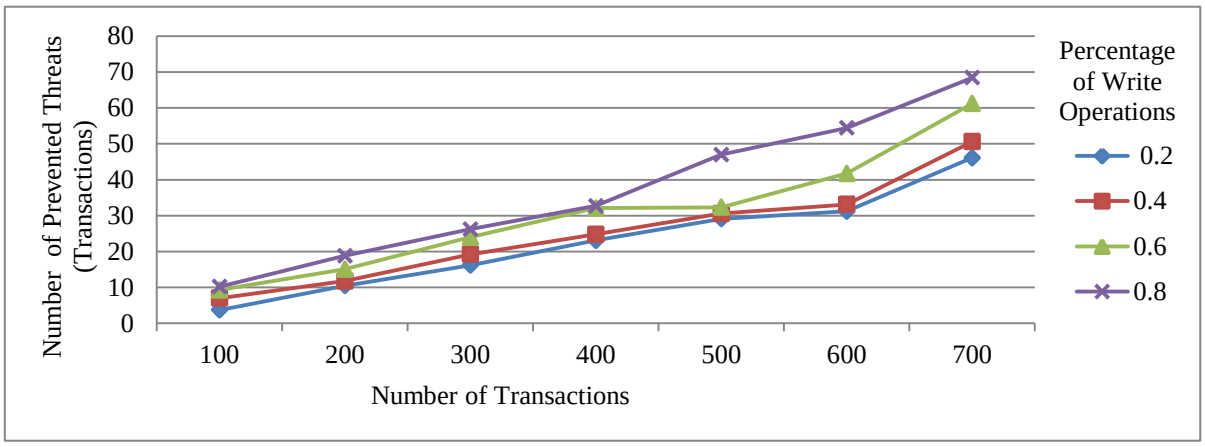

Figure 5.9. Number of Prevented Threat (risky transactions) vs. Number of Transactions and the Percentage of Write Operations

Figure 5.9 shows the number of prevented threats (risky transactions) according to the number of transactions and the percentage of write operations. The number of insiders used in the simulation is 20, and W_a and W_s are 2 and 4 respectively. Obviously, the figure shows that as the number of transactions increases, the number prevented threats, which is a trivial result since the number of threats is directly proportional to the number of transactions increases. In addition, the figure shows that, for the same number of transactions, the number of prevented threats increases

as the percentage of write operations increases. This is an expected result since increasing the number of write operations in a transaction maximizes the probability of modifying sensitive data items, which in turn increases the possibility of threat (and the prevented threat using the approach). As discussed earlier, the system chooses the solution with the lowest cost to prevent a threat.

Preventing a threat is not always the best solution. In some cases, allowing insiders to access unauthorized data is better than impeding the tasks of insiders. The solutions in these cases depend on the weights associated with availability and sensitivity, and on the sensitivity of data items as well. Figure 5.10 shows the ratio of the number of prevented threats to the number of allowed threats according to a variable ratio between W_a and W_s. For instance, when $W_a : W_s$ is 1:4, the number of prevented threats is about 49 times more than the number of allowed threats. The figure shows that the number of prevented threats is greater than that of allowed threats when $W_a < W_s$, and the value (Number of Prevented Threats / Number of Allowed Threats) increases as W_a gets smaller. Whereas, the number of allowed threats is greater than the number of prevented threats when $W_a > W_s$ and the value (Number of Prevented Threats / Number of Allowed Threats) decreases as W_s gets smaller.

Obviously, the simulation demonstrates that the proposed approaches prevent insider threat efficiently taking into account systems preferences, where systems have to choose between breaching the security and limiting the availability of data items.

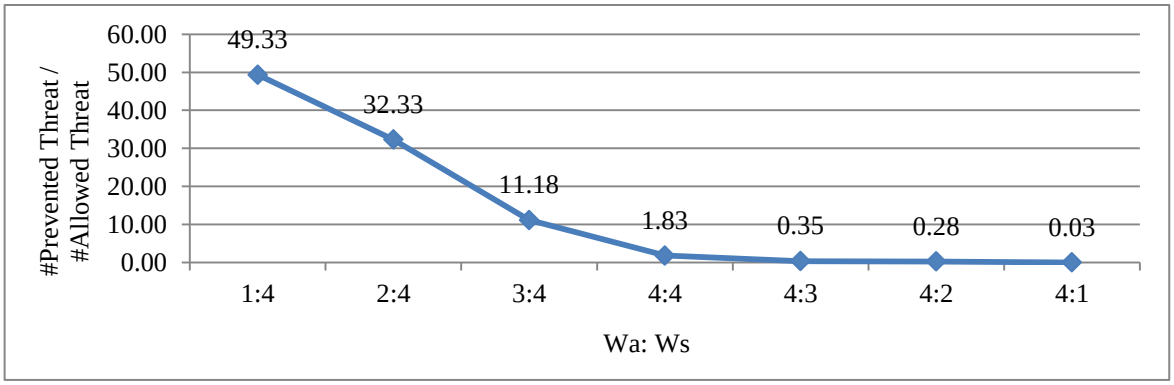

Figure 5.10. The Number of Prevented Threat / the Number of Allowed Threat according to Different Ratios of Wa:Ws

6. Organizing Access Privileges: Preventing Insider Threat without Affecting the Availability of Data Items

6.1 Introduction

A task of an insider may consist of several operations and may need access to different data items. The operations on data items of a task form a partial order. That is, some operations on data items should be performed in some specific order, while other operations can be performed without any order among themselves. In some cases, the order of granting access to data items in order to execute a task determines the level of risk. In other words, different orders of accesses to data items imposes different levels of risks. This chapter discusses the importance of organizing operations in concurrent tasks. Moreover, it demonstrates how to organize accesses to data items such that insider threat is prevented without affecting the availability of data items.

6.2 The Importance of Organizing Accesses to Data Items

The history of data accesses by insiders, when combined with data access requests, may pose a serious threat. As discussed in previous chapters, insiders can use data items they have accessed in the past (in knowledgebases) to infer sensitive information. In concurrent tasks, not considering knowledgebases and random executing of tasks' operations may pose threat or limit insiders' tasks. However, the operations of tasks can be organized such that the threat of knowledgebases is eliminated without limiting insiders' tasks. Figure 6.1 shows an instance of the task of an insider, which is represented by a task graph, and his/her knowledgebase. A task graph is defined as follows.

Definition 30 (Task Graph). *A task graph TG(V,E) is a directed graph that is used to show the operations and their precedence constraints in a task, where:*

- *V represents operations on data items, such that:*
  - *r indicates read access.*
  - *w indicates write access.*
- *E represents edges, such that:*
  - *An edge e(O(x),O(y)) means that O(x) should be executed before O(y), where O is an operation, and x and y are data items.*

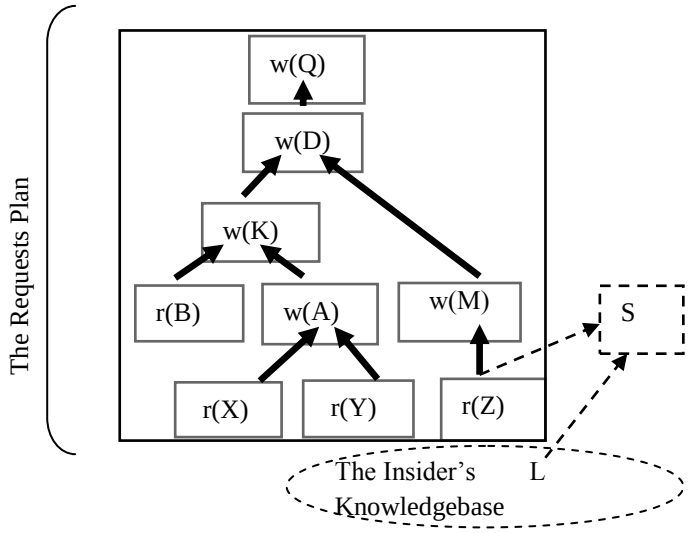

Figure 6.1. A Sequence of Operations to Perform a Task.

Obviously, a task graph represents the data items that should be accessed to perform a task as well as the required operations (read or write). In addition, it demonstrates the required sequences of accesses on those data items (precedence constraints). Moreover, tasks' graphs show the different sequences of operations that can be followed to execute the same task. Strictly speaking, operations that have no precedence constraints can be executed in different orders to perform the task. In Figure 6.1, the task graph is enclosed by rectangles and the knowledgebase

is represented by a dashed oval. Solid arrows represent precedence constraints between operations. Whereas dashed arrows point to unauthorized inferred knowledge. For example, *L* and *z* can be used to get information (using dependencies) about *S*, which is unauthorized to the insider under consideration. Hence, the dashed square contains unauthorized data items.

In order to execute the insider's task, the insider needs to get access to data items X and Y to work on the data item A. Next, s/he needs an access to B to work on K and so on until s/he reaches Q. Notice, that the insider can get access to Z before getting access to X or Y. That is, s/he can work on M before working on A and vice versa because the two operations are independent. However, s/he must work on A before working on K since they are dependent.

As discussed earlier, the dashed arrows indicate that the insider can use the data item L, which is in his/her knowledgebase, along with the data item Z to get unauthorized information about the data item S. If the system discovers this threat and denies the insider's request to Z, the availability of data items will be limited, which degrades the performance of the system. On the other hand, if the system does not discover this threat, unauthorized information will be exposed, which breaches the security of the system. To prevent the threat without limiting the insider's task, the insider should not get a read access to Z until the lifetime of L expires. However, delaying the insider's job until the lifetime of L expires is not a good solution always, since the delay may be too long. To solve this problem, the insider is given access to other data items to work on an independent operation(s) until the lifetime of L expires. For instance, the insider may get access to X and Y to work on A first before s/he get access to Z. We can force the lifetime of L to expire by giving other insiders who want to modify L write access on it. Thus, the lifetime

of L expires after it is updated by those insiders. In this case, after the insider under consideration finishes his/her work on A, giving him/her access to Z does not pose threat. Obviously, this approach enables concurrent insiders to get the accesses they need on different data items without breaching the security of the system. This example shows the importance of organizing accesses to data items in preventing insider threat and preserving the availability of data items.

The terms *risky data item* and *risky request* are used in the rest of the chapter to indicate data items that can be combined to get unauthorized information. Definitions 31 and 32 formally define these terms.

Definition 31 (A Risky Data Item). *Given the knowledgebase(KB) of an insider, where KB= {k_1, …, k_n}, k_i is called a risky data item if it can be used with other data items, which can be requested by the insider, to acquire unauthorized information.*

Definition 32 (A Risky Request). *A request O_{ij}, which indicates the i^{th} request in the j^{th} task that belongs to insider j, is called a risky request if it can be combined with a risky data item in j's knowledgebase to acquire unauthorized information.*

For instance, in the previous example, the data item Z, when it is requested by the insider, is called a risky request and the data item L in his/her knowledgebase is called a risky data item. Organizing accesses to data items is applied by considering all concurrent insiders and their tasks, the data items and operations required for each task, and the dependencies between operations. It must be noted that investigating the knowledgebase of each insider is a major part.

Obviously, the system should have a full knowledge about the tasks that concurrent insiders are planning to execute. However, insiders execute tasks in two ways: as a batch of operations or as one operation at a time. The tasks in the latter case are called *undeclared tasks*, which are defined formally as follows.

Definition 33 (Undeclared Tasks). *A task* $S = \{O_1, O_2, \ldots, O_n\}$, *where* $O_1, O_2, \ldots, O_n$ *are operations on data items, is called an undeclared task if it is sent by an insider to be executed as one operation at a time.*

Clearly, the operations of an undeclared task are sent by insiders to be executed as one by one, where each operation is executed in a single transaction. However, when a task is sent as one transaction that contains all the operations needed by the task, the task is called a *declared task*. Section 6.3 discusses the methods of organizing accesses in declared tasks, while section 6.4 discusses organizing accesses in undeclared tasks.

6.3 Organizing Operations in Declared Tasks

In this type of tasks, an insider sends a task's operations in one transaction. Thus, the system has full knowledge about the task the insider is planning to do. This enables the system to organize the operations of concurrent tasks (before granting risky requests) in a good sequence, which preserves the availability and security of data items.

To understand how to organize accesses, consider the relational database schema in Figure 6.2. The database has the dependency {Rank → Base_Salary}. Assume that the data items (Name,

Rank), (Rank, Base_Salary), and (Name, Experience) are insensitive information, while the data items (Name, Base_Salary) and (Name, Total_Salary) are sensitive information. In addition,

Professor Table

ID	FName	LName	Rank	Experience	DeptID
20012	James	White	Assistant Prof	3	168
20013	Bob	Tailor	Full Prof	2	597
20014	Sami	Gibson	Associate Prof	5	168

Rank_Salary Table

Rank	Base_Salary
Assistant Prof	100K
Associate Prof	120K
Full Prof	140K

Department Table

DeptID	Name	Location
168	Computer Science	SSED
597	Electrical Engineering	LKEF

Figure 6.2. A Part of Academic Staff Database

suppose that the salary of an academic staff is computed using the formula: *Total_Salary = Base_Salary + 200 * Experience.* Now, assume that there are two insiders who are concurrently working on their tasks. Both insiders are not allowed to get information about sensitive data items. The task of the first insider (Insider1) consists of the following queries.

Query 1: "Retrieve the name and the rank of all computer science professors"

```
Select P.FName, P. LName, P.Rank
From Professor P, Department D
Where P.DeptID = D.DeptID and D.Dname = "Computer Science"
```

Query 2: "Retrieve the experience of professor Sami Gibson"

```
Select P.Experience
From Professor P
```

Where P.ID = 20014

Query 3:"Retrieve the Base_Salary of associate professors"

Select R.Rank, R.Base_Salary

From Rank_Salary R

Where R.Rank="Associate Prof"

Obviously, the result that is added to the knowledge of *Insider1* if s/he is granted the privilege to execute *Query1* is: (< James White, Assistant Prof >, < Sami Gibson, Associate Prof >). Similarly, if *Insider1* is granted a privilege to execute *Query2* and *Query3*, s/he will have the information (< Sami Gibson, 5 >) and (< Associate Prof, 120K >) respectively. Now, suppose that the task of the second insider (*Insider2)* consists of the following query.

Query 4:"Promote Sami Gibson to a Full Prof"

Update table Professor

Set Rank = 'Full Prof'

Where ID=20014

Assume that the queries *Query1* and *Query2* are executed successfully. Thus, *Insider1* has the rank and the experience of "Sami Gibson" in his/her knowledge. Next, if *Query3* is executed, *Insider1* gets the knowledge < Associate Prof, 120K >. In this case, s/he can combine this insensitive knowledge with the insensitive knowledge < Sami Gibson, Associate Prof > and <Sami Gibson, 5> to get the unauthorized information <Sami Gibson, 120K> and <Sami Gibson, 121K>, which indicates the Base_Salary and Total_Salary of "Sami Gibson". Although

executing the task of *Insider2* after that changes this information, *Insider1* still knows that at the time s/he executed his/her query Sami's total salary was 121K, which is correct and unauthorized information. On the other hand, if the system discovers this threat and prevents *Insider1's* request, *Insider1's* job will be rejected. Thus, both cases affect the system negatively.

Let us consider another scenario for satisfying the requests of the two insiders. Suppose that *Query4* is executed before *Query3*. This means that *Insider2* promotes the rank of "Sami Gibson" from associate professor to full professor before *Insider1* gets access to the *Base_Salary* of associate professors. In this case, if *Insider1* uses the data item in his/her knowledgebase to infer information, his/her inference will be incorrect. Thus, *Insider1*'s task will be executed normally. Obviously, this scenario prevents insider threat without limiting the availability of data items. Notice that the data item <Name, Rank>, which has been acquired by *Query1* is called a risky data item. Similarly, the request <Rank, Salary> that is requested by *Query3* is called a risky request. The request in *Query4* (updating the rank of "Sami Gibson") is called an *Effacement Request* since it removes the threat of the risky request in *Query3*. An effacement request is defined as follows.

Definition 34 (An Effacement Request). *A request O_{ij}, which indicates the i^{th} request in the j^{th} task that belongs to insider j, is called an effacement request if it satisfies the following conditions:*

1- $O=Write(R) \wedge (Write(R) \rightarrow (Expire(R) =True))$.

2- $(R \in KB(h)) \wedge (j \neq h)$.

Definition 34 states that an effacement request by an insider j must be a write operation that updates a data item R and makes it expire, where R belongs to the knowledgebase of a different insider than j. That is, a write request by an insider j that updates a data item in his/her knowledgebase is not considered an effacement request.

The example clarifies the importance of choosing the order of executing the requested operations in preventing insider threat without limiting insiders' tasks. It shows that the sequence <Query1, Query2, Query3, Query4> pose a threat, while the sequence <Query1, Query2, Query4, Query3> does not pose any threat. The first sequence is called a *safe sequence*, which is introduced next in Definition 35. The next section discusses how to choose a safe sequence for executing the operations of concurrent tasks.

Definition 35 (A Safe Sequence). *Given a sequence of operations* $S = \{O_1, O_2, \ldots, O_n\}$, *where* $O_1, O_2, \ldots, O_n$ *are operations on data items that belong to concurrent tasks. S is called a Safe Sequence if executing the operations in S's order does not reveal unauthorized information and preserves the availability of data items.*

6.3.1 Choosing a Safe Sequence

After considering concurrent insiders in a system and their tasks, the system organizes the accesses to data items to prevent any insider threat and preserve the availability. There are many possible sequences of data access to execute a task. Finding a safe sequence is the objective of this section. However, choosing a safe sequence of operations is not always achievable. Thus, in these cases, an acceptable sequence should be chosen, which is defined as follows.

Definition 36 (Acceptable Sequence). *Given a sequence of operations S = {O_1,O_2,…, O_n}, where O_1,O_2,…, O_n are operations on data items that belong to concurrent tasks. S is called an Acceptable Sequence if executing the operations in S's order reveals insignificant unauthorized information, which does not pose any intolerable threat to the system, and preserves the availability of data items.*

Security administrators decide whether the revealed information is insignificant or not, or whether it poses an intolerable threat. In order to choose either a safe or an acceptable sequence, the risk of granting each request is computed. The risk of a request is the maximum difference between the sensitivity of each data item that may be revealed by granting the request and the threshold value of the insider about that data item. The following formula shows how to compute the risk of requested operations, where R_j is a request by insider I, n is the number of data items in the database under consideration, Sensitivity(d_i) is a the amount of information that may be revealed about the data item d_i by granting R_j, and Threshold (I, d_i) is the threshold value of I about d_i.

$$\text{Risk}(R_j) = Max_{i=0}^{n}(\text{Sensitivity}(d_i) - \text{Threshold}(I, d_i)) \quad (1)$$

Formula 1 measures the risk of a request independently. That is, it looks at the knowledgebase of the insider under consideration to see if the current request can be combined with some data items in his/her knowledgebase to get unauthorized information. The formula does not pay attention to other operations that are executed before it. However, the risk value of a sequence is

computed by considering the order of operation in the sequence. The risk of the each sequence is computed using Formula 2, where n indicates the number of requests in a sequence.

$$Risk(S_j) = \Sigma^{n}_{i=1} Risk(R_i|R_1 \ldots R_{i-1}) \quad (2)$$

Obviously, the value that is computed in Formula 2 is the sum of the risk values of requests in a sequence with taking into account the order of requests. To clarify this point, we should mention that the risk of a request that is computed independently (the value computed in Formula 1) differs when we consider previous requests. That is, as we discussed before, a request(s), say R, may update a risky data item(s), say K, that exist in the knowledgebase of an insider and make it expire. Thus, the insider who has K in his/her knowledgebase cannot use it with his/her risky requests to infer unauthorized information. This action may reduce the risk values of successor risky requests after R in the corresponding sequence that can be combined with K to infer unauthorized information, which may reduce the risk value of the corresponding sequence.

Using this method, a safe or acceptable sequence is chosen, which poses the lowest risk among different sequences. We should mention here that choosing a good sequence may be limited by the fact that some operations are dependent on each other. That is, some operations must be executed before other operations. Moreover, when producing a safe sequence, operations that should be executed before effacement requests should be put before them in any sequence.

6.3.2 Limitations and Possible Solutions

Organizing accesses to data items either eliminates or significantly reduces the threat of a risky request demanded by an insider. As discussed earlier, this is performed by letting other insiders

modify risky data items so that they are expired before they can be used with risky requests to launch an attack. But what can be done if there is no insider who requests write accesses to risky data items? To solve this problem, the granting of a risky request may be delayed until an effacement request is made. However, this method would result in data unavailability and degrading systems performance. Moreover, if the insider must get access to the requested data item to perform his/her job on a timely manner, the mentioned solution is unacceptable.

When delay is unacceptable, an incorrect value of the risky request can be granted to the insider. After that, the system corrects the results based on the correct value of that risky request. When incorrect values of data items are provided to insiders, they will not be able to infer correct values of dependent data items. We propose to do so when the inferable data is sensitive. However, this approach may affect insiders' trust about the system. To mitigate this issue, incorrect but close enough values must be provided while making sure that the values still do not disclose any sensitive data. To know how much information one can infer, the Neural Dependency and Inference Graph (NDIG) is used. An example of NDIG is shown in Figure 6.3, where cyclic inference edges are omitted for simplicity.

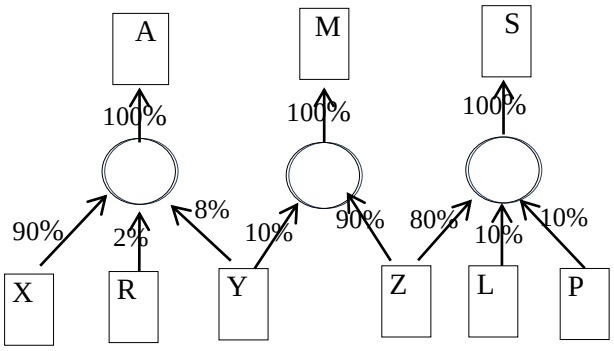

Figure 6.3. A Part of an NDIG of a Database

For example, suppose that an insider K had accessed the data items L and P in this database. Later, s/he requested the data item Z. Figure 6.4 shows K's task and knowledgebase. Assume that K's threshold is 100% for all data items except for the sensitive data item S, which is 65%. In addition, assume that the value of S ranges between 0 and 100, and it is computed using the formula: S = 4*Z + L+ P.

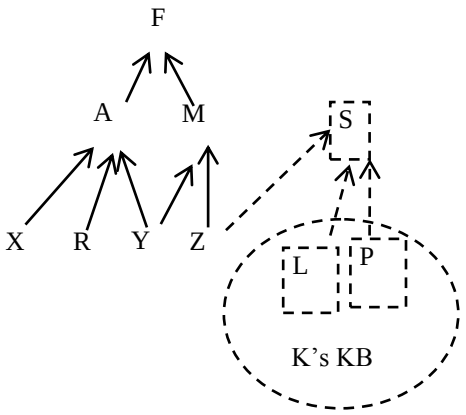

Figure 6.4. Insider K's Task and Knowledgebase

Obviously, using the proposed approach, the insider is given an access to the data items X, R, and Y to work on A first. S/he is not given access to risky request Z because he/she can combine it with the risky data items L and P, which are in his/her knowledgebase, to get information more than the allowed about the data item S. Suppose that at the time, there is no insider requesting a write access on either L or P. In addition, assume that due to the time sensitive nature of insider K's task, the system has to grant him/her the access to data item Z. Clearly, granting the request poses a threat. Thus, to avoid this threat, the insider is given an incorrect value of Z. Notice that the given incorrect value does not mean that the value of Z is changed in the database. It means

that the system provides an incorrect value to the insider. However, this incorrect value should satisfy two conditions, which are:

a) It should not be very different from the correct value, otherwise this would affect the insider's trust on the system if the insider has a guess on the range of the value.
b) Using the value, the user should not be able to make a correct estimation of the sensitive data item.

After giving the insider an incorrect value of Z, the system should track the subsequent modifications on the data items that the insider K makes using the incorrect value of Z and correct those using the right value of Z. This process applies to other insiders who access such damaged data items as well. Damage assessment and recovery are not the focus of the dissertation and, therefore, the methods will not be discussed here. As a reference, interested readers may review the work presented in [Yalamanchili04]. Notice that giving incorrect values may pose work overhead to trace the changes and fix the affected data items. However, this could be much less costly than breaching system security or rejecting insiders' tasks. Moreover, fixing affected data items can be performed when systems have less work overhead.

6.3.3 An Example Scenario

Let Table 6.1 represent the set of concurrent insiders in the system as demonstrated in Figure 6.2, their knowledgebases and their current requests of data items. The dependencies in the database are shown in Table 6.2. Table 6.3 shows the data items in the system and their sensitivity values. In addition, it shows the threshold values of data items according to the insiders. The amount of information an insider can get about a data item is computed depending on the NDIG of the corresponding database.

Table 6.1. Insiders and their Knowledgebases and Requests

Insiders	Knowledgebase	Requests
U_1	K_1 = < James White, Assistant Prof > K_2 = < Sami Gibson, Associate Prof > K_3 = < Sami Gibson, Experience = 5 >	$R(q_1)$ = Read(<Rank, Base_Salary> of Associate Prof)
U_2	–	$W(q_2)$ = Write(<Base_Salary> of Full Prof) $W(q_3)$ = Write(<Rank> of Sami Gibson)

Table 6.2. Dependencies

No.	Dependency
1	<Name,Rank>,<Rank,Base_Salary>, <Name,Experience> → <Name, Total _Salary>
2	<Name,Rank>,<Rank,Base_Salary> → <Name, Base_Salary>

Table 6.3. Sensitivity and Threshold Values of Data items

Data Item	Sensitivity	Threshold U_1	Threshold U_2
K_1	20%	100%	50%
K_2	20%	100%	40%
K_3	10%	100%	100%
q_1	50%	70%	70%
q_2	70%	100%	100%
q_3	80%	100%	100%
<Name, Total _Salary>	100%	10%	0%
<Name, Base_Salary>	100%	20%	10%

Obviously, all requests are independent. Notice that both requests R_1 and R_2 read and write the *Base_Salary* respectively. However, R_1 read the *Base_Salary* of associate professors, which is different from R_2 that updates the *Base_Salary* of full professors. Thus, they are independent. This offers flexibility in organizing them in a good manner to form the lowest risk sequence. Clearly, there are six possible sequences for granting the requests, which are:

- $S_1 = \{ U_1.R(q_1), U_2.W(q_2), U_2.W(q_3) \}$
- $S_2 = \{ U_1.R(q_1), U_2.W(q_3), U_2.W(q_2) \}$

- S_3= { $U_2.W(q_2)$, $U_1.R(q_1)$, $U_2.W(q_3)$ }
- S_4= {$U_2.W(q_2)$, $U_2.W(q_3)$, $U_1.R(q_1)$ }
- S_5= {$U_2.W(q_3)$, $U_2.W(q_2)$, $U_1.R(q_1)$ }
- S_6= {$U_2.W(q_3)$, $U_1.R(q_1)$, $U_2.W(q_2)$ }

Notice that the data items requested by insiders are allowed to both insiders. However, the data item in request q_1 can be combined with risky data items in U_1's knowledgebase to infer unauthorized information. The risk of each independent request is computed using Formula 1 as follows.

- *Risk* ($U_1.R(q_1)$) = 100% -10% = 90%.
- *Risk* ($U_2.W(q_2)$) = 0%.
- *Risk* ($U_2.W(q_3)$) = 0%.

Notice that negative Risk values are considered 0. The risk of each sequence is computed using formula 2 as follows.

- *Risk* (S_1) = *Risk* ($U_1.R(q_1)$) + *Risk* ($U_2.W(q_2)$ | $U_1.R(q_1)$) + *Risk* ($U_2.W(q_3)$ | $U_2.W(q_2)$, $U_1.R(q_1)$) = 90%+0%+0%=90%.
- *Risk* (S_2) = 90%.
- *Risk* (S_3) = 90%.
- *Risk* (S_4) = 0%.
- *Risk* (S_5) = 0%
- *Risk* (S_6) = 0%.

Notice that in sequences S_4, S_5 and S_6, the second insider U_2 updates the Rank of "Sami Gibson" to full professor before the first insider U_1 gets access to the *Base_Salary* of associate professors. Thus, the inference that U_1 makes about the salary of "Sami Gibson" is wrong, which means that these sequences are safe. However, the situation is different for the rest of sequences. In these sequences, U_1 deduces correct information about the *Base_Salary* and *Total_Salary* of "Sami Gibson". Obviously, the system can choose one of the sequences S_4, S_5 and S_6 to grant accesses to the insiders. Hence, the system does not need to compute the risk values for all possible sequences when there a safe sequence exists, which can be produced directly by placing effacement requests before risky requests in the sequence.

6.4 Organizing Operations in Undeclared Tasks

The method that was proposed in the previous section is applicable when tasks are declared. Strictly speaking, that approach depends on the assumption that systems are familiar with all operations of concurrent insiders' tasks. Thus, the approach can check all concurrent tasks' operations and produce a safe sequence when threat is discovered. However, this approach fails when tasks are undeclared. In this case, systems are familiar with the operations that are launched before discovering threat, but not all operations. Thus, the proposed approach in the previous section cannot be used to produce a safe sequence in this case.

This section develops methods that can predict and prevent insider threat without limiting the availability of data items in concurrent undeclared tasks. In order to achieve this goal, models are proposed to predict the complete operations of undeclared tasks when threat is discovered. Then, the predicted tasks are organized into a safe sequence.

6.4.1 Predicting the Complete Operations of Undeclared Tasks

As discussed earlier, preventing insider threat without limiting the availability of data items is performed by organizing all operations of tasks into a safe sequence. Thus, the complete operations of undeclared tasks should be predicted in order to produce a safe sequence.

Figure 6.5 shows the predicting process of the original tasks (complete operations) of undeclared tasks. The predicting is needed when a threat alert is raised while concurrent undeclared tasks are being executed. The alert occurs when an insider orders a risky request. At this point, the system has only the operations that have been executed before raising the alert. These operations are called *partial tasks* of the original tasks. Next, the partial tasks are compared to a set of *training tasks*, which are a set of daily tasks that are normally executed in the system. The training tasks set can be developed in two ways. First, it can be developed during the building of the system itself by addressing all possible tasks that will be executed in the system. Second, it can be developed using the tasks that exist in the log file.

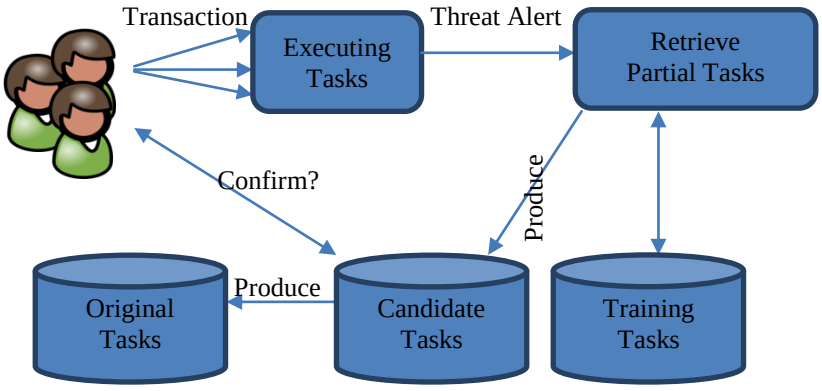

Figure 6.5. Predicting Undeclared Tasks

The purpose of comparing partial tasks to training tasks is to retrieve a set of candidate tasks to the partial tasks. The candidate tasks contain the correct (complete) tasks of the undeclared tasks

(partial tasks). Retrieving candidate tasks is performed as follows. For each partial task *t*, each task in the set of training tasks that starts with the same operations (from the left) of *t* is retrieved as a candidate task for *t*. For instance, the task $S = \{ r(x), r(y), w(a), r(b), w(k)\}$ is retrieved as a candidate task for the partial task $Ps=\{r(x), r(y), w(a)\}$.

Retrieving all candidate tasks needs considering all possible ways of executing tasks. A task can be executed in different ways. Insiders who have a specific role access the same data items to perform a specific task. However, the order of accesses to data items to perform the task may differ from one insider to another, which is due to the fact that a task can be executed using different orders of its operations. For instance, suppose that the "*Salary*" of an academic staff is computed as follows: *Salary = Base_Salary + 200 * Experience*. Now, to update the *Salary* of a professor who finishes another year of experience, the task would be as follows. {*Read(Base_Salary), Write(Experience), Write(Salary)*} or {*Write(Experience), Read (Base_Salary), Write(Salary)*}. These two sequences of executing the task are called *patterns* of the task. Notice that the patterns show that there is a precedence constraint between {*Read(Base_Salary) ,Write(Experience)}* and *Write(Salary)*, but there is no precedence constraint between *Read(Base_Salary)* and *Write(Experience)*. Thus, the latter two operations can be executed in different orders. The following theorem states the conditions of considering two tasks as patterns for a task.

Theorem 1. *Given the two tasks:* $S = \{Os_1(ds_1), Os_2(ds_2),\ldots, Os_n(ds_n)\}$ *and* $X = \{Ox_1(dx_1), Ox_2(dx_2),\ldots, Ox_m(dx_m)\}$, *where (Os, ds) and (Ox,dx) indicates operations (read or write) and data*

items that belong to tasks S and X respectively. S and X are considered two patterns for the same task if the following conditions are satisfied.

1- n = m.

2- $\forall d: d \in S \Leftrightarrow d \in X$.

3- $\forall d: O(d) \in S \Leftrightarrow O(d) \in X$.

4- $\forall (O=Write): (Osi(dsi) \rightarrow Osj(dsj)) \Leftrightarrow (Oxi(dxi) \rightarrow Oxj(dxj))$.

Theorem 1 states that two tasks are considered patterns for the same task if they have the same number of data items, the same operations on the same data items, and the same order of write operations.

Retrieving all candidate tasks requires checking all patterns of tasks. For instance, the task {*Read(Base_Salary), Write(Experience), Write(Salary)*} may not exist in the set of training tasks in this form. Instead, it may exist in the form:{*Write(Experience), Read (Base_Salary), Write(Salary)*}. However, the task is still retrieved as a candidate task for the partial task {*Read(Base_Salary), Write(Experience)*}. Notice that retrieved original tasks should be confirmed by insiders to avoid executing incorrect tasks. The next section fully details this approach.

6.4.2 Preventing Insider Threat and Preserving the Availability in Undeclared Tasks

As discussed earlier, the purpose of constructing training tasks is to discover the tasks of undeclared tasks, and then, produce a safe sequence that prevent insider threat without limiting insiders' tasks. The prediction is required when an insider sends a risky request. The prediction

method is run for all concurrent insiders' tasks. When the process is finished, a safe sequence can be produced. The next section discusses this process in details.

6.4.2.1 The Algorithm

Algorithm 6.1 shows how to predict the full tasks of undeclared tasks and produce a safe sequence. The algorithm works as follows. When an insider, say *K*, sends a risky request, the algorithm starts predicting the candidate tasks of all undeclared tasks that are running concurrently. The prediction is based on the previous operations that are executed before discovering the risky request (steps1-2). For each insider, say *Z*, the prediction is performed by comparing the previous operations that are executed by *Z* to the training tasks (step 3). As discussed earlier, the training task(s) that starts (from the left) by the same operations, with taking into account the patterns of tasks, is retrieved as a candidate tasks(s) for Z's undeclared task (step 4). After retrieving candidate tasks, their operations are organized in a safe sequence. To produce a safe sequence, the algorithm searches for an effacement request for the risky data item insider K's knowledgebase (step 7 and step 13). Then, the algorithm organizes the operations of tasks by executing the effacement request (and its precedent operations) before executing the risky request to prevent the possible threat (steps 8-9 and steps 15 - 19). Notice that, in step 25, if no effacement request exists, the algorithm denies the risky request to prevent the threat. Hence, the precedent operations represent the operations that have to be executed before the effacement request due to operations dependencies.

Algorithm 6.1 Preventing Insider Threat

Input: The set of concurrent insiders R={r_1 … r_n}, the set of concurrent tasks of insiders T={t_1 … t_n}, knowledgebases, the set of training tasks TT={Tt_1..Tt_z}, Safe Sequence SS ={}, operations dependencies, Candidate Tasks set CT ={}, Risky Request (Rq), Risky Data Item (RD).

Assumptions: Training tasks that are stored in the system represent all tasks that are normally executed in the system.

Output: A safe sequence for executing undeclared tasks' operations.

1. **For** each insider $r_i \in R$ // *when a risky request Rq is discovered*
2. Retrieve r_i's previous requests L ={q_{i1}… q_{ix}} *//operations executed before the risky request*
3. Let G = {Tt_1… Tt_s} the set of training tasks that have *L* as starting operations // *with taking into account the patterns of tasks*
4. CT_i = G // *retrieve G as a candidate task(s) for r_i's undeclared task*
5. **If** $|CT_k|$ =1 for all insiders $r_k \in R$ *// one candidate task for each undeclared task*
6. Correct _Task = CT_k // *Retrieve CT_k as the correct task for r_k's undeclared task*
7. Search for an effacement request (ER) *//effacement request that updates the risky data item*
8. **If** ER Exists
9. SS = {$O_{i=1}$, $O_{i=2}$, … , $O_{i=n}$}: index (ER)< index(RR) //*organize the operations of tasks so that ER(and its precedent operations) is executed before the Rq*
10. **Else** go to step 25
11. **Else if** $|CT_k| > 1$ for an insider r_k's undeclared task*// more than one candidate task?*
12. Assume that the risky request $Rq \in t_p$, where t_p is the task of r_p*// risky request*
13. Search for an effacement request (ER)
14. **If** ER Exists
15. Let ER $\in Tt_c$ be an effacement request, where $Tt_c \in CT_f \wedge f \neq p$ *//effacement request*
16. Suggest ER and its precedent operations to insider r_f *//executing the effacement request before the risky request*
17. **If** r_f accepts *ER*
18. Executes ER and its precedent operations *//correct prediction of the f's undeclared task*
19. Execute Rq *//since the threat is eliminated*

20. Allow insiders to continue performing their tasks normally *//after eliminating the threat*

21. **If** another Rq is requested *//another threat shows up*

22. Repeat the steps 1 to 20

23. **Else**

24. Search for another ER and repeat steps 13-22*// incorrect candidate task*

25. **Else if** there is no ER exists *//no effacement operations exist*

26. Deny Rq

The following example demonstrates how the algorithm works. Suppose that there are three insiders R_1, R_2 and R_3 who are executing their tasks concurrently. Assume that the knowledgebase of insider R_1 ($KB(R_1)$) contains the data item *p*, which can be combined with the data item *d* to infer the unauthorized information S_1 using the dependency$\{(d, p) \rightarrow S_1\}$. Thus, the data item *p* is a risky data item. Now, assume that R_1 requests the data item *d*, which is in this case a risky request. Using step 1, the algorithm looks back at the previous requests that have been executed by the insiders before discovering the risky request. Suppose that the previous requests are as follows: $[R_1: r(a), r(b), w(c), \underline{r(d)}]$, $[R_2: r(x),r(y),r(z)]$ and$[R_3: r(m), w(n)]$.Then, the algorithm searches for candidate tasks that start with these operations (steps 2-4). Suppose that the graphs of the retrieved candidate tasks are as shown in Figure 6.6. The double circle around a request indicates an effacement request, while a circle indicates a risky request. Since there are more than one candidate task that have been retrieved for an undeclared task, the algorithm moves to step 11. Next, the algorithm checks which candidate task contains the effacement request (steps 12-15). In this example, the effacement request is *w(p)*, which is in the first candidate task of insider R_2*'s* undeclared task. The algorithm suggests this request (and its precedent operations) to insider R_2 (step 16).

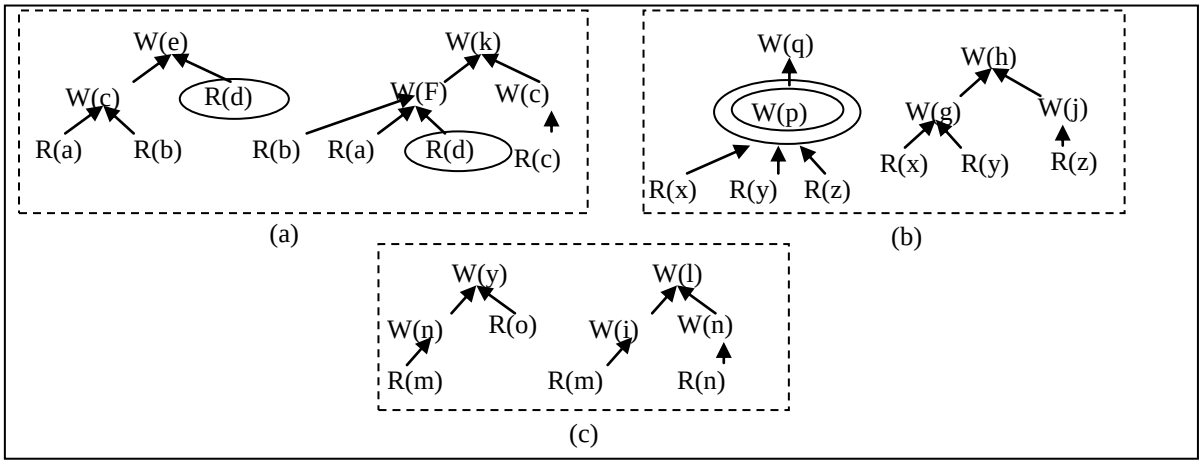

Figure 6.6.The Graphs of Candidate Tasks for the Undeclared Tasks of Insiders R_1,R_2 and R_3 (a) The Candidate Tasks Graphs for R_1's Task (b) The Candidate Tasks Graphs for R_2's Task (c) The Candidate Tasks Graphs for R_3's Task

If insider R_2 accepts and executes this request (steps 17- 18), the algorithm grants insider R_1 his/her risky request *r(d)* to be executed after the effacement request (step 19). The executed sequence of operations until this point is called a *partial sequence*. Next, the execution of the operations of tasks is performed normally as they are requested by insiders until another risky request appears. If the proposed effacement request *w(p)* is rejected by insider R_2, this means that the predicted task is incorrect. Thus, the algorithm searches for another effacement request (steps 23-24). If no effacement request exists, R_1's task is rejected to prevent the threat.

Notice that the algorithm completely organizes the operations of candidate tasks when one candidate task is retrieved for each undeclared task (steps 5-10). It assumes that the candidate tasks are the correct tasks. Whereas it produces a partial sequence (as in the example) when there is more than one candidate task retrieved for an undeclared task (steps 11-24). In the latter case, the algorithm does not organize all the operations of predicted tasks in a sequence. Instead, it stops after executing the effacement request and the risky request respectively. Then, it enables

insiders to continue executing their operations normally. This choice is to eliminate the possible overhead that may arise when predicting an incorrect task from the candidate tasks, which requires re-predicting and reorganizing tasks operations.

In real scenarios, applying the algorithm could pose time overhead. However, not using the approach limits the availability of data items (insiders' tasks) or degrades a system's security. Strictly speaking, not discovering risky requests exposes systems' sensitive information, which poses dangerous effects on critical systems such as military systems. Similarly, discovering risky requests and denying them to prevent insider threat limits the availability of data items and affects insiders' tasks, which degrades systems performance especially in online systems. Contrarily, adopting the proposed approach prevents insider threat without affecting insiders' tasks. The system may decide which is least costly according to its own requirements? Exposing system's sensitive assets or limiting the availability of data items or accepting the delay of applying the proposed approaches. Section 6.5 demonstrates the conditions under which the proposed approach works with greatest performance and least delay.

6.4.2.2 The Proof of Correctness

The algorithm consists of two main parts, which are predicting correct candidate tasks and organizing the operations of predicted tasks in a safe sequence. The following theorems prove that the algorithm addresses the correct steps to perform these parts.

Theorem 2*. Given the training tasks* $Tt_1, Tt_2 \ldots Tt_n$ *, and a poset of operations* $S = \{O_1, O_2 \ldots O_k\}$*:* $S \subset UT$*, where UT represents an undeclared task, then:*

$Tt_i \in CT(UT) \Leftrightarrow S \subset Tt_i \wedge Tt_i \in patterns(UT) \wedge (\forall_{z=0}^{k} (S(z) = Tt_i(z)))$

Where S represents the set of operations that are executed to perform UT before discovering a risky request, CT(UT) indicates the set of candidate tasks for UT.

The theorem addresses the conditions of considering a training task as a candidate task for an undeclared task.

Proof:

The proof of this algorithm is fairly straight forward. Since *UT* is an undeclared task and the only known part of it is *S*, any training task, say Tt_i, that starts with the operations of *S* could be the correct task of *UT* (steps 2-4). In other words, the insider who has executed the operations in *S*, is probably going to perform the training task Tt_i. Thus, Tt_i is considered a candidate task for *UT*. Notice that the comparison is performed with taking into account the precedence constraints and the patterns of tasks as discussed in 4.2 (step 3).□

Theorem 3*. Given two sequences for executing the operations of concurrent tasks in a system, S(O_1…, ER,…,RR,…, O_n) and S'(O_1,…, RR,…,ER,…, O_n), where O_i is an operation, RR is a risky request and ER is an effacement request. Then:*

- *Threat (S) < Threat(S').*
- *Availability (DI(S)) >Availability(DI(S')), where DI(S) and DI(S') indicates the data items in S and S' respectively.*

The theorem states that executing an effacement request before a risky request prevents insider threat and preserves the availability of data items. Choosing a safe sequence is performed using steps 8-9 and 12-20.

Proof:

Suppose that *RR, ER* and *RD* are a risky request, an effacement request, and a risky data item respectively. As discussed earlier, based on definitions 31 and 32, combining *RR* and *RD* may expose unauthorized information, which is threat. On the contrary, if the system discovers this threat when an insider requests *RR*, the system may deny *RR*, which reduces the availability of the data items needed for the insider's task. Thus, both possibilities are problematic. Next, the correctness of the algorithm is proved by contradiction as follows.

(Proof by contradiction) Suppose that both sequences *S* and *S'* are executed, but *Threat(S) < Threat(S')* and *Availability(DI(S)) <Availability(DI(S')).* Now, in sequence *S*, *ER* is executed before *RR*. This means that *RD* is expired before executing *RR*. Thus, the inference that is based on combining *RD* with *RR* is incorrect, which means that no threat exists. Moreover, *RR* is granted and executed, which means that the availability of data items is not limited. Meanwhile, in *S', RR* is executed before *ER*. In this case, the inference that is based on combining *RR* and *RD* is correct since the value of *RD* is not expired yet, which is threat. Thus, *Threat (S') > Threat(S)*, which contradicts the assumption. Contrarily, suppose that the system discovers this possible threat and denies the insider access to *RR* to prevent the threat. This action limits the availability of *RR*, which is a data item in *S'*. As a result, *Availability(DI(S)) >Availability(DI(S'))*. Contradiction.□

The theorems above prove the correctness of predicting tasks graphs using the algorithm. Moreover, they prove the correctness of organizing access privileges in preventing the threat and increasing the availability of data items.

6.4.3 A Real World Example Scenario

The following example clarifies how the algorithm works in real world scenarios. Suppose that the two insiders Amy and Ashley want to submit the following tasks to the corresponding system. Hence, the insiders submit their tasks as one operation at a time. Amy's task is as follows.

Query1:

Select E.Address
From Employee E
Where E.Name="Jif"

Query2:

Select E.Rank
From Employee E
Where E.Name="Jif"

Where Ashley's task is as follows.

Query3:

Select S.BaseSalary
From Employee S
Where S.Name="Jif"

Query4:

Update table Employee
Set Experience= 5
Where Name= "Jif"

Query5:

Update table Employee
Set Salary =BaseSalary +
*100*Experience*
Where Name= "Jif"

Suppose that the corresponding relational database has the following dependencies: *{Rank →BaseSalary}*, *{Salary → (BaseSalary, Experience)}*, where *Salary = BaseSalary + 100* Experience*. In addition, assume that Amy's knowledgebase contains the information (Jif, 4),

which indicates the experience of Jif, and the information (Jif, Salary) is unauthorized information to Amy. Now, suppose that *Query1* and *Query3* have been executed and Amy has just submitted *Query2*, which contains a risky request *Read(Rank)*. This request invokes the algorithm. In this case, the algorithm starts predicting the candidate tasks for these undeclared tasks. Assume that the algorithm has retrieved one candidate task for each task (steps 1-4) as shown in Figure 6.7. In this case, the algorithm supposes that these are the correct candidate tasks for the undeclared tasks (Steps 5-6). Step 7 searches for an effacement request, which is *w(Experience)*(Query4) in this case, and step 8 organizes the operations in a safe sequence by placing *w(Experience)* before *Read(Rank)*. At the end, the safe sequence would be as follows: *{r(Address), w(Experience), r(Rank), r(BaseSalary), w(Salary)}.*This sequence removes the possible threat that may arise if Amy's task is executed before Ashley's task. Moreover, the availability of data items is preserved and both tasks are executed.

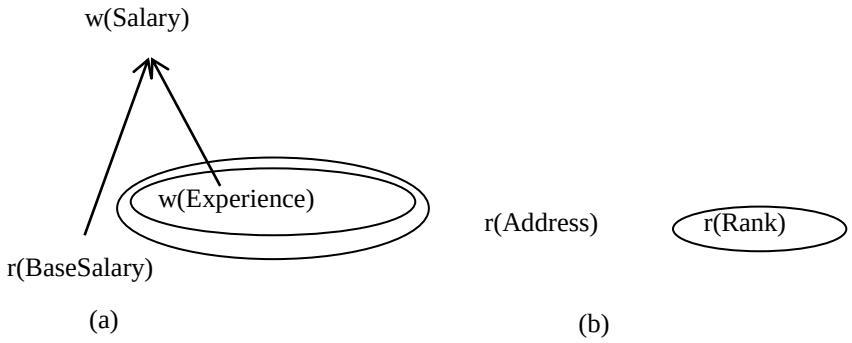

Figure 6.7. (a) Amy's Task Graph (b) Ashley's Task Graph

6.5 Experiments and Analysis

The simulation was performed using SQL Server and MS C#.net to test the effectiveness of the proposed approaches. A sample relational database of 10 tables was created manually. The

dependencies and access permissions were created randomly. Similarly, different sizes of training tasks were created randomly as well.

The simulation consists of two parts. The first part demonstrates the percentage of prevented threat using a safe sequence under different conditions. It shows the effectiveness of the proposed approaches discussed in section 6.3 (declared tasks). The second part demonstrates the relationship between the positions of risky requests in risky transactions and the number of retrieved candidate tasks. Moreover, it shows the relationship between the size of training sets and the number of retrieved candidate tasks. This part shows demonstrates the effectiveness of using the proposed approaches discussed in section 6.4 (undeclared tasks).

6.5.1 The Percentage of Safe Sequences

In this part, the simulation parameters consist of the number of concurrent insiders and the percentage of write operations in transactions. For the same parameters' values, the simulation was executed many times and the average was taken as the result. Similarly, the simulation was performed according to different sizes of training sets, and the average was taken as the final result. We should mention here that all risky transactions were caught and prevented using the proposed approaches. As discussed earlier, the proposed approaches prevent threat by choosing a safe sequence, or denying insiders' risky requests if finding a safe sequence (effacement request) is not possible. The simulation shows the percentage of prevented threat using a safe sequence in comparing to overall prevented threat. Producing a safe sequence prevents threat without limiting the availability of data items. However, denying access requests prevents threat but limits the availability of some data items. Hence, it is assumed that the system is able to find

a safe sequence when an effacement request exists. Figures 6.8 and 6.9 show the results of this part of simulation.

Figure 6.8 shows the percentage of safe sequences using the proposed approach when the number of concurrent insiders is variable. The percentage of write operations in transactions is fixed at 50% in this simulation. As shown, the percentage of prevented threat is 0 at round 1. This is expected since at round 1 the insiders start executing transactions. Thus, their knowledgebases are empty, which means that there is no threat posed by their knowledgebase, and as a result, there is no prevented threat. As knowledgebases grow, the threat may increase, but the figure shows no trend such as increasing or decreasing in the percentage of safe sequences. The analysis of this result is as follows. When the knowledgebase of an insider gets larger, the probability of finding a risky data item and a risky request may increase. However, the number of risky data items or risky requests by an insider is limited by the maximum number of data items s/he can request to perform his/her task, which is set to be 6 for this simulation. Moreover, there are many insiders in the system that may order effacement requests. This makes the percentage (Safe Sequences/ Prevented Threats) stable in general as knowledgebases grow.

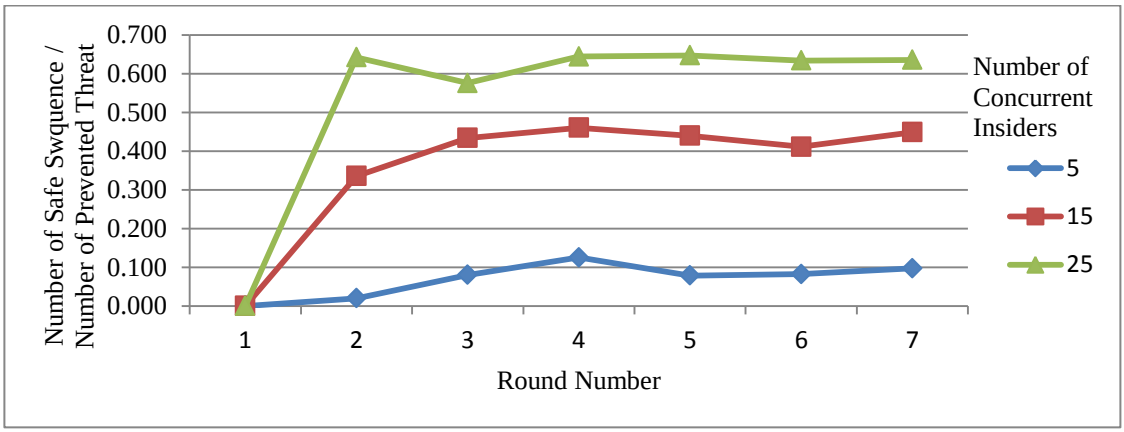

Figure 6.8.The percentage of the prevented threat using a safe sequence in comparing to different numbers of concurrent insiders.

The figure shows that the probability of finding a safe sequence increases when the number of concurrent insiders increases. Clearly, increasing the number of concurrent insiders may increase the number of risky requests. But the number of risky requests that may be demanded by an insider is limited as discussed earlier. Whereas the number of effacement requests increases as the number of concurrent insiders increase. Thus, the probability of finding an effacement request (and a safe sequence) gets larger when the number of concurrent transactions increases.

Figure 6.9 shows the percentage of prevented threat using a safe sequence according to different percentages of write operations. The number of concurrent insiders is fixed at 10 for this simulation. The figure shows that when the percentage of write operations increases, the percentage of finding a safe sequence increases. Obviously, increasing the write operations in the concurrent transactions increases the possibility of finding effacement requests, which increases the possibility of finding a safe sequence. Moreover, the figure shows that no increasing or decreasing trend exists as the knowledgebases of insiders grow, which is the same result that is shown in Figure 6.8.

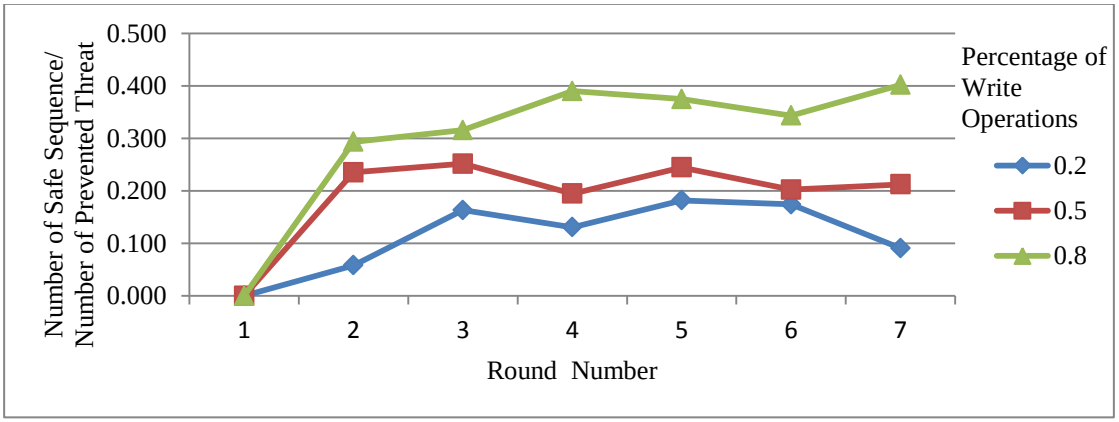

Figure 6.9. The percentage of the prevented threat using a safe sequence in comparing to different percentages of write operations

As discussed earlier, finding a safe sequence to run transactions is better than denying accesses according to both security and availability viewpoints. The simulation shows that the probability of finding a safe sequence gets larger as the number of concurrent insiders and the percentage of write operations in transactions increase; the percentage of safe sequences reaches about 65% when the number of insiders is 25 (with fixed write percentage = 50%), and it reaches about 40% when the percentage of write operations is 80% (with fixed number of concurrent insiders = 10). We should mention here that the number of posed threat depend also on the threshold values of insiders about data items. Hence, insiders are assumed to have direct access permission to 30% of data items in this simulation.

6.5.2 Retrieved Candidate Tasks

Predicting candidate tasks may pose delays in executing insiders' tasks. The delay time depends on the number of candidate (similar) tasks that may be retrieved when a risky request is encountered. This part of simulation was performed to show how much delay is needed to retrieve the correct candidate task of an insider's task when using the proposed approach in section 6.4 (undeclared tasks). The simulation was performed according to different sizes of training tasks. Figure 6.10 shows the relationship between the number of retrieved candidate tasks and the position of the risky request in a risky transaction. The figure shows that the number of candidate tasks increases when the position of the risky request (RR) gets smaller. This is a normal result since when a risky request is encountered in a transaction; the approach looks back into the operations that are executed before the risky request in the transaction. Then, it searches for tasks in the training set that start with the same set of data items, operations and the order of write operations. Thus, when the number of these data items increases, the number

of tasks that satisfy these conditions decreases. That is, when the position of RR gets greater, the number of candidate tasks decreases, and vice versa.

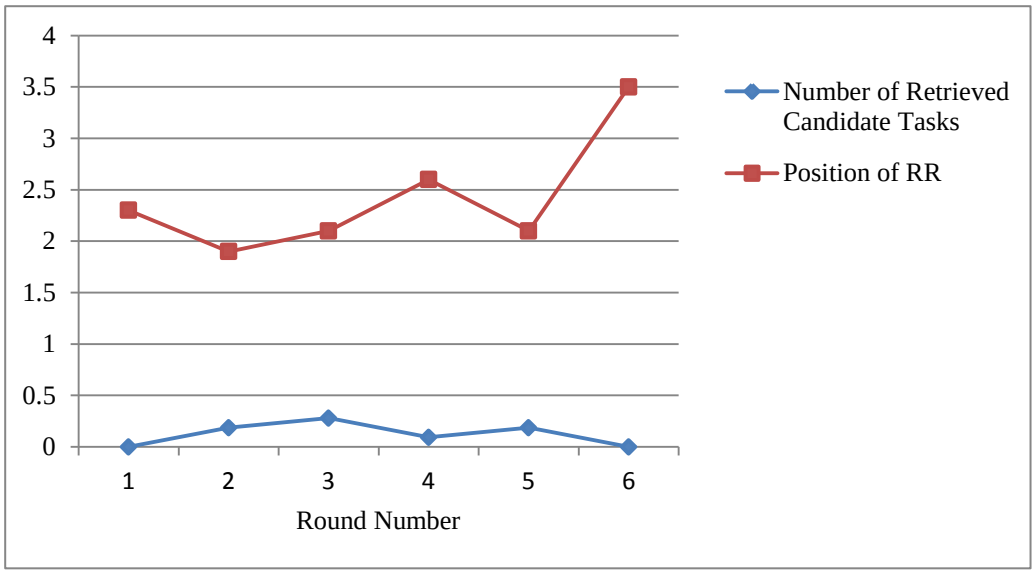

Figure 6.10.The relationship between the number of retrieved candidate tasks and the position of a risky request (RR) in a risky transaction.

The figure shows that the maximum number of retrieved candidate tasks for each insider is about 0.28, which happens when the position of RR is 2.1. These values are average values since the simulation was performed for several rounds and the average was taken as the final result. This number of candidate tasks poses little cost on delay when executing transactions, especially, when this cost is compared to security or availability costs that would be paid in case of insider attacks. However, this simulation was performed with 15 concurrent insiders and 150 training tasks. The next figure reveals more details when these numbers get larger.

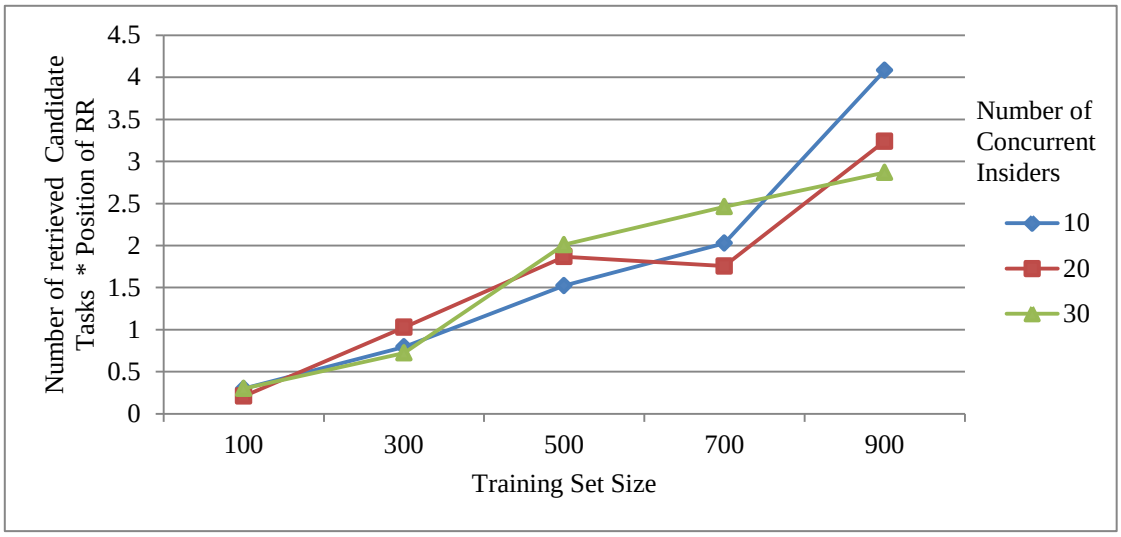

Figure 6.11. The relationship between the number of retrieved candidate tasks and the size of the training set.

Figure 6.11 shows the relationship between the number of candidate tasks and the size of the training set according to different number of concurrent insiders. We should mention here that the number of candidate tasks represents the average number for each insider, which has been calculated according to several rounds. The figure shows that as the size of the training set increases, the number of candidate tasks increases, which is a fairly straight forword restult. Notice that, in y-axis, the number of candidate tasks is multplied by the postion of RR. This conversion is to measure the number of candidate tasks according to the same postion of RR. For instance, suppose that when the size of the training set is 300, the number of candidate tasks and the position of RR are 1 and 2 respectively. Similarly, assume that when the size of the training set is 500, those values are 0.5 and 3 respectively. To measure the number of candidate tasks when the position of RR is 1 in both sizes of training sets, 1 is multiplied by 2 and 0.5 is multiplied by 3. Notice that this conversion is performed by multiplication and not by division since the postion of RR and the number of candidate tasks is inversely correlated.

As shown in the figure, the number of retrieved candidate tasks for each insider is about 0.3 when the training set size is 100, but it reaches about 4 when the training set size is about 900. Thus, the proposed approach works better when the number of daily transactions in a system gets lower. The figure shows that there is no relationship between the number of concurrent insiders and the number of candidate tasks. This is a normal result since the number of retrieved candidate tasks in this simulation is the average for each insider. Strictly speaking, the simulation was performed in several rounds, and the total number of retrieved candidate tasks is divided by the number of rounds and the number of concurrent insiders in the system. Thus, as the number of insiders increases, the total number of candidate tasks increases, but the average stay the same in general.

7. Tackling Insider Threat in Cloud Relational Database Systems

7.1 Introduction

Using virtual machines to run applications is one of the main features of using the cloud, where cloud platforms host many applications (tenants). Adopting *multi-tenancy* reduces the operating cost by allowing powerful resources sharing among tenants. Managing virtual machines are required in order to achieve affective recourses utilization. Load balancing are performed using *live migration* [Das11], where virtual machines are migrated from overloaded nodes to idle (or low-loaded) nodes. However, live migration may pose a delay in delivering services since it limits the availability during migration process. Developing methodologies for efficient and low cost live migration has got significant attentions by researchers. A number of methods have been proposed for effective live migration such as Albatross [Das11] and Zehpyr [Elmore11].

Security is one of the major concerns when moving to the cloud. Proving the security of data in the cloud is mandatory to achieve users' trust of cloud providers. Multi-tenancy could be a vulnerability source. For instance, an insider may use shared resources to breach the security of other insiders' tasks [Takabi10]. Moreover, the guarantee of protecting data that resides on the cloud from the threat of cloud providers' employees is a major requirement by customers. Encryption is one of the methods suggested to protect data. For instance, CryptDB, Homomorphic Encryption (HOM) and Encryption Deterministic (DET) are encryption methods that can execute the operations of relational databases queries on encrypted data [Curino11]. These methods prevent the cloud providers' employees from exposing users' data even when customers' queries are executed. Besides protecting data, authentication of users is another major

concern when moving to cloud. Thus, the development of digital identity management systems is crucial for cloud computing [Bertino09]. The agreements between cloud customers and cloud providers regarding the security and offered services are set using Service Level Agreements SLAs, which should be maintained by cloud providers.

Insider threat is one of the problems that worry organizations and individuals about cloud computing. Moving data into the cloud increases the number of insiders, which increases insider threat. Moreover, preventing data in the cloud from insiders may require new methodologies different from those used to protect data stored locally. This chapter discusses insider threat at cloud relational databases (cloud RDBMS). To the best of our knowledge, this is the first work that tackles this problem at cloud RDBMS. As discussed earlier, knowledgebase is a serious source for insider threat. Insiders can combine the data items that they accessed (in their knowledgebases) with other data items that they can request to infer sensitive information. Cloud RDBMS has new vulnerabilities that may enable insiders to breach the existed solutions and launch attacks using their knowledgebases. One of these possible vulnerabilities is the migration (live migration) of insiders' tasks across availability zones and data centers due to load balancing. This chapter shows how existing insider threat preventing methodologies, which prevent insiders from exploiting their knowledgebases to pose threat, can be breached by insiders in cloud RDBMS. Moreover, it proposes three models that can be used to prevent the threat of knowledgebases in this new environment. Furthermore, it addresses the conditions under which they can be used effectively. In addition, it discusses how to manage the effect of updating data items in knowledgebases using the proposed models.

7.2 Insider Threat in Cloud Relational Databases

Cloud providers store data in multiple datacenters that are both geographically and logically separated. A datacenter consists of connected servers and storage systems. Storage systems are aggregated into storage pools to form logical storage, which can be accessed from different computer systems that share the storage pool. One of the key benefits of this feature is that data can be replicated or moved to other locations (storage locations) transparently to servers using it [Stryer12].

Availability zones in each datacenter are connected via inexpensive and low latency network. To achieve greater performance and fault tolerance, an application's traffic may be distributed across multiple availability zones and data centers, which is called *elastic load balancing* [Amazon12a]. Figure 7.1 shows the structure of Amazon cloud services. The figure shows that Amazon has five data centers across the globe. Each datacenter has more than one availability zone (AZ) [Amazon12b].

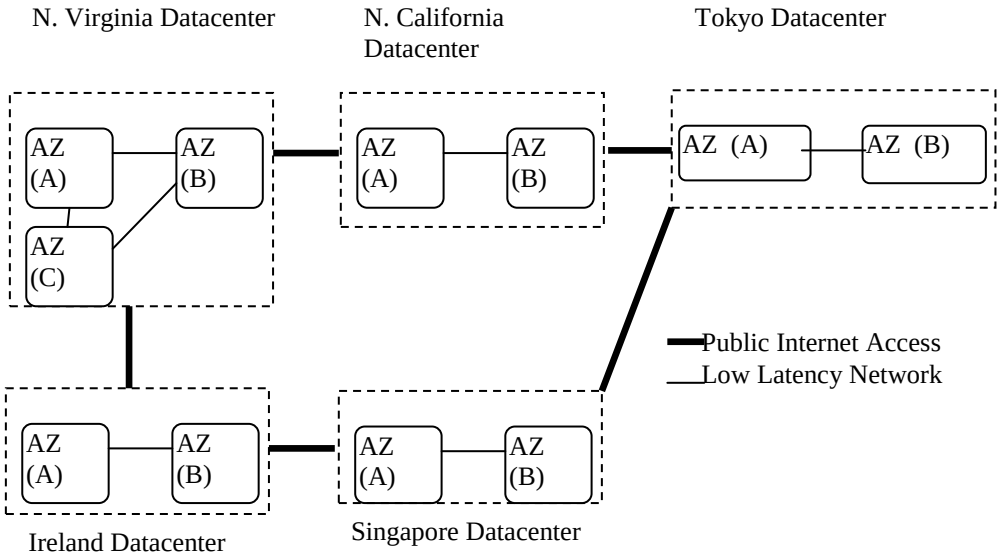

Figure 7.1. Amazon's Cloud Structure

Cloud relational databases are fragmented and replicated to increase availability and reliability. Replication of data across availability zones and datacenters should be consistent. Workloads on replicas' nodes are balanced using *live migration*, where tenants (applications) are migrated from overloaded nodes to idle (or low-loaded) nodes to achieve *load balancing*. Users have no control on choosing the location or the instance that they prefer. Cloud systems choose the server, the location and the storage that are needed for executing a process depending on some criteria such as the amount of load on servers or availability zones. Thus, different user's requests may be executed on different instances in the same availability zone or in different availability zones or data centers.

Replication and load balancing increases the performance of cloud relational database systems. However, it may increase the probability of insider threat. Such a threat arises when a cloud relational database system fails to use the knowledgebase of insiders to detect threat. In other words, an insider may combine data items s/he gets from database instances in different availability zones to pose threat. Figure 7.2 shows the problem. The insider accesses the data item D_1 in the availability zone $1(AZ_1)$ (that may have the knowledgebase of the insider) and then accesses D_2 in the availability zone n (AZ_n) (that may not have the insider's knowledgebase). In this case, the system on the availability zone n fails to detect this threat and enables the insider to access D_2. Thus, the insider combines the two data items and gets the sensitive information S_1, which is a threat.

In addition, insiders are allowed to access a cloud from any site on the globe, which is one of the features that the cloud offers. Cloud systems connect insiders to the closest availability zone (if it

is not overloaded) to execute their queries in order to achieve the best performance. This means that insiders may be connected to different availability zones when they travel and work from different sites. In this case, insiders may be able to launch attacks using the same scenario described in Figure 7.2. We should mention here that to the best of our knowledge no research has discussed the threat of knowledgebase in cloud environment, and no research has addressed how to manage knowledgebases in this new environment.

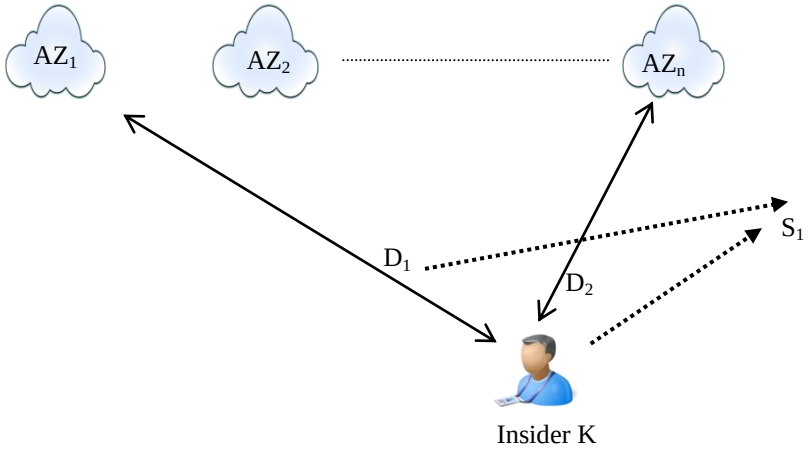

Figure 7.2. Insider Threat in Cloud RDBMS

In light of the previous discussion, an up-to-date knowledgebase should be available to be checked at each insider's access for all insiders to prevent insider threat. Furthermore, the knowledgebase of an insider should be updated after each access an insider performs. Thus, cloud RDBMS needs new methodologies to build, store and synchronize knowledgebases in cloud environments since local knowledgebase are no longer suitable.

7.3 Mitigating the Threat of insiders' knowledgebases

Securing cloud RDBMS against insider threat needs a methodology that monitors the activities of insiders in different instances and locations of cloud relational databases. The knowledgebases of insiders should be checked and synchronized to achieve this purpose. In traditional RDBMS, building, maintaining, and checking knowledgebases are the responsibilities of organizations (owners). Nonetheless, when moving to the cloud, these operations are transferred to cloud providers (Cloud RDBMS). Keeping these responsibilities for local systems when moving to the cloud violates the concept of cloud computing. Moreover, keeping the knowledgebase of an insider in local storage needs transferring it with every access by the insider, which is an infeasible way due to the network overhead that it poses especially when knowledgebase gets large. This section introduces three frameworks to maintain knowledgebases in Cloud RDBMS, and demonstrates the features and limitations of each one.

7.3.1 Peer-to-Peer Model

In this model, the knowledgebase of each insider is built and stored in all availability zones. At each access of an insider to a data item in an availability zone, the knowledgebase of the insider in the availability zone is updated. Next, the updates are sent to all other availability zones and data centers simultaneously to keep knowledgebases consistent. Transactions are monitored locally at each availability zone or database instance. Thus, insider threat monitoring is performed locally without a need to communicate with other nodes. Figure 7.3 shows the proposed framework, where *AZ* denotes an availability zone, *LB* denotes load balancing and *U(KBs)* denotes updating of knowledgebases. As shown in the figure, an insider sends his/her query to a cloud RDBMS. The cloud system sends the query to the closest availability zone. If

the availability zone has a high load, the query is transferred to another availability zone. In both cases, the insider's knowledgebase is checked to ensure that there is no threat. Once the query is executed, the knowledgebase of the insider is updated and all replicas of the knowledgebase in all other availability zones are updated as well.

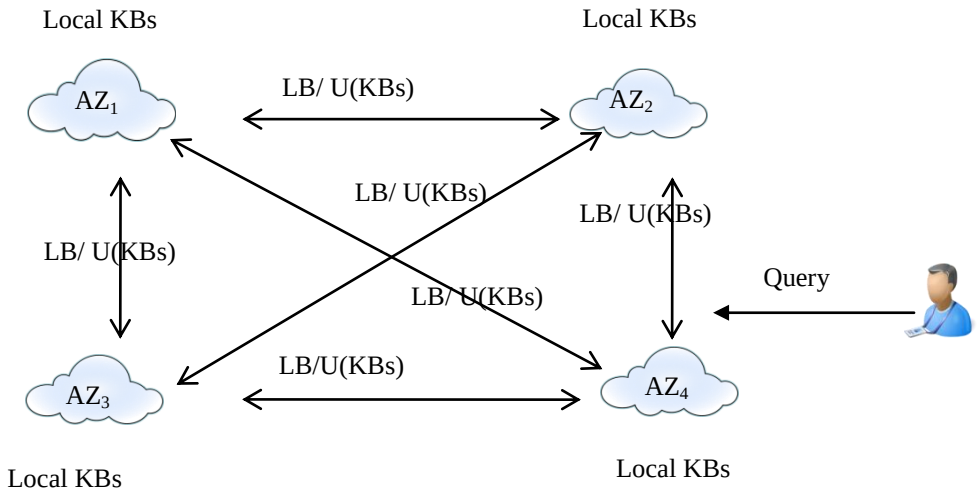

Figure 7.3. Peer-to-Peer Model

A key benefit of this model is that there is no single point of failure. Moreover, transactions and threat detection processes are executed fast since all processing are performed within a single availability zone and no communications are needed with other zones of the cloud system. Furthermore, the processing needed for manipulating knowledgebases are distributed among all availability zones, which balances the load on them. The challenge that arises when using this model is the profiling of activities (building knowledgebases) for each insider. Local profiling is faster in processing transactions, but it imposes synchronization problems. Knowledgebases in all database servers should be updated simultaneously. Otherwise, insiders may access different data items in different sites (due to load balancing) and combine them using dependencies to pose threat as discussed earlier. Keeping knowledgebases updated needs a lot of immediate

processing, which is both time and resources consuming, and it causes delays in processing transactions. In summary, this model poses high network traffic and delays transaction processing especially in case of large number of replicas. Therefore, this approach is suitable when the number of instances is small.

To enhance the performance of this model, updating knowledgebases in some availability zones can be postponed when the processing load or network traffic is high. In this case, new processing requests by insiders should be distributed between up-to-date availability zones only. Other availability zones can be updated when traffic is low. This helps in increasing the performance of processing transactions and in decreasing the delay that may be caused in case of high network traffic.

7.3.2 Centralized Model

This model uses a coordinator site that builds, stores, and manages the knowledgebases of all insiders. Moreover, each insider's query is sent to the coordinator first. Then, the coordinator checks the query against insider threat using the insider's knowledgebase that it has. If no threat exists, the coordinator sends the query to one of the cloud RDBMS nodes (in an availability zone) with taking into account the load balancing. After executing the query successfully, the cloud RDBMS sends back an acknowledgement to the coordinator so that it can update the knowledgebase of the insider. The model in this case has a bottleneck. That is, failure of the coordinator turns down the entire system of insider threat prediction and prevention. Figure 7.4 shows the modified model. The modified model uses a secondary coordinator to mitigate the bottleneck problem, which is similar to the idea used in damage recovery in distributed systems

by Zue and Panda [Zue04]. The secondary coordinator is used only in case of failure. However, the secondary knowledgebase should be updated to keep both knowledgebases consistent as shown in Figure 7.4, where *U(KBs)* indicates updating the knowledgebases, *LB* indicates load balancing, and "*Ackn.*" denotes an acknowledgement.

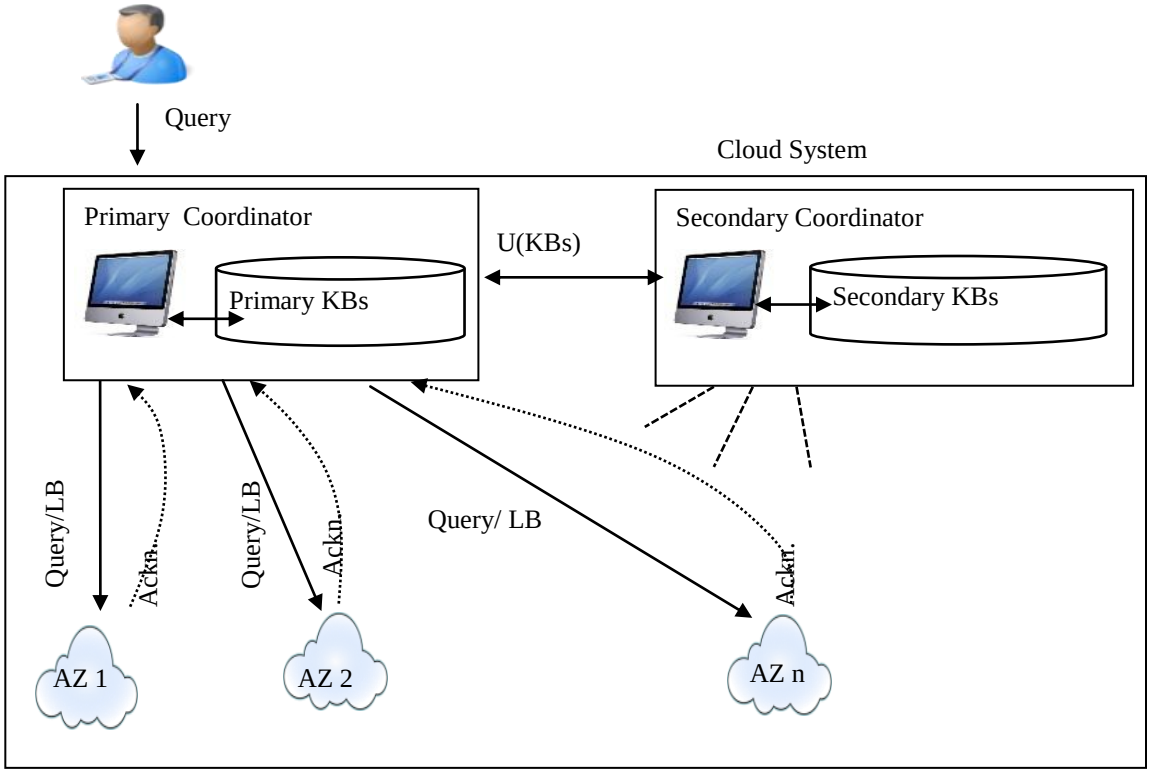

Figure 7.4. Centralized Model

The advantages of this model include relatively small amount of network traffic in compared to the previous model. Thus, this model is more scalable than Peer-to-Peer model. Moreover, the updates are synchronized between the instances of knowledgebases only (the primary and secondary sites). That is, no delay occurs because of the synchronizing process between knowledgebases instances. However, the delay may happen because all requests are inspected

and filtered at the central unit. Therefore, the central unit should be equipped with high performance capabilities to carry out this job.

7.3.3 Mobile-Knowledgebases Model

This model has the advantages of Peer-to-Peer model, and mitigates its disadvantages. In this model, an availability zone in a data center stores knowledgebases of insiders who are geographically close to it, instead of storing knowledgebases of all insiders. For example, Figure 7.5 shows how knowledgebases of insiders in the USA may be stored; where Arkansas insiders' knowledgebases can be stored in availability zone 4, and Washington insiders' knowledgebases can be stored in availability zone 1. Hence, availability zones may belong to different data centers. This model depends on the assumption that insiders are highly probably performing most of their work in one location (i.e. a company complex). However, an insider may perform his/her work from different (geographically) locations, which is a key advantage of

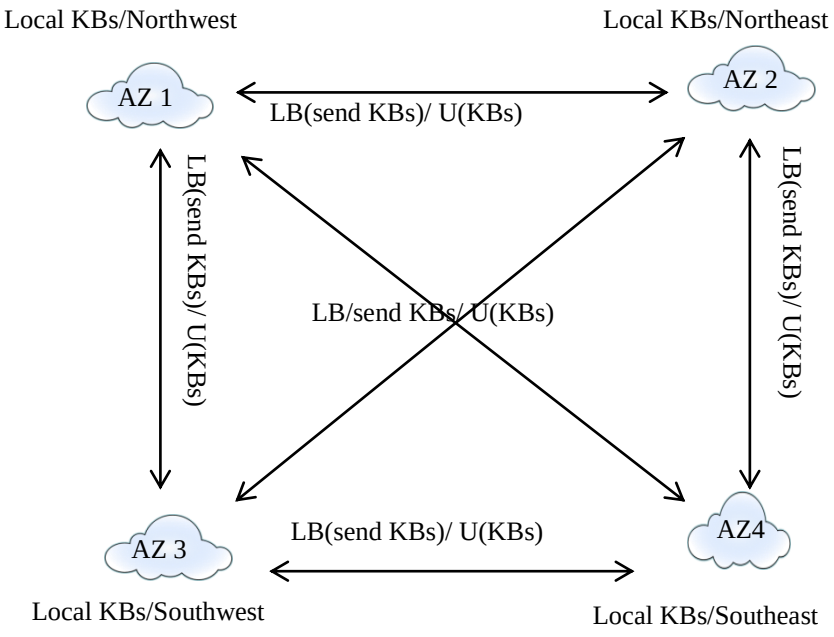

Figure 7.5. An Example of Mobile Knowledgebases Model

cloud computing. In this case, the cloud system should send a copy of the knowledgebase of the insider to the new location to check his/her queries against insider threat. In the figure, *Send KBs* stands for sending a copy of a knowledgebase of an insider, which may be needed when balancing a load or when an insider accesses an availability zone that does not have his/her knowledgebase.

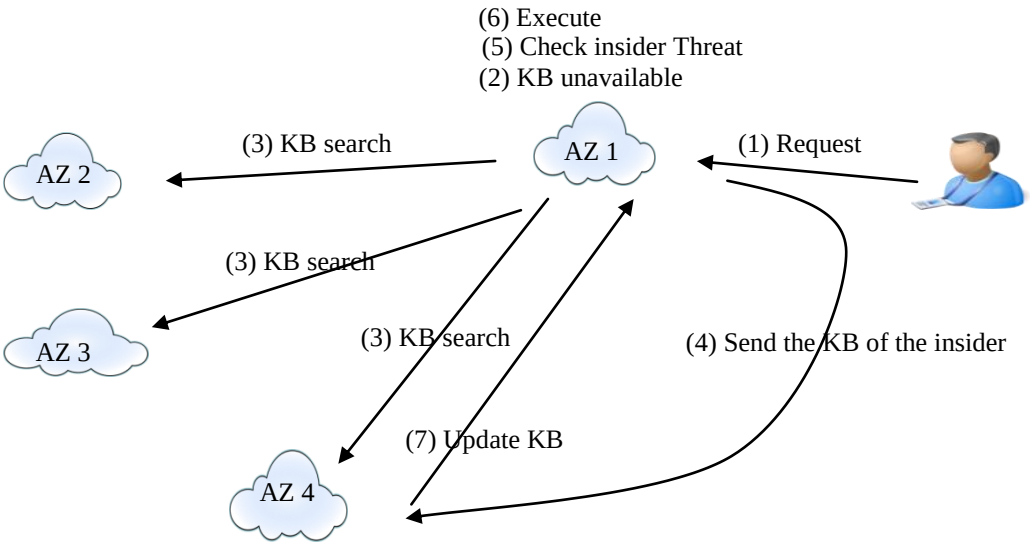

Figure 7.6. Executing Queries in a Mobile-Knowledgebases Model

To show how the model works, suppose that an insider, say Bob, works for a company in Arkansas, which belongs to availability zone 4 (AZ4). Assume that Bob travelled to Washington, which belongs to availability zone 1 (AZ1), and he wants to perform some work for his company. Figure 7.6 shows how the model works in this case. Bob sends his query to the cloud system, which forwards his request to AZ1. The cloud system in this availability zone checks whether Bob's knowledgebase exits or not. Since the knowledgebase is not available, the cloud system in AZ1 contacts other availability zones asking for the knowledgebase of Bob.

Availability zone 4, which has the knowledgebase, sends a copy of Bob's knowledgebase to AZ1. Then, the cloud system in AZ1 checks whether there is a threat posed by Bob. If there is no threat, AZ1 executes Bob's request and sends the updates on Bob's knowledgebase to AZ4. Algorithm 7.1 shows how this model works in details.

Algorithm 7.1. Executing Transactions and Insider Threat Prevention in Mobile-Knowledgebases Model

Input. Dependencies, Knowledgebases
Output. Updated Knowledgebases, Insider Threat Prevention

```
1. For each insider K
2.   Store an instance of the knowledgebase of K (KB(K)) in the closest AZ
3. For each transaction T submitted by an insider F to AZ J
4.    Check the availability of KB(F) in J
5.    If KB(F) exists in J
6.       Retrieve KB(F) and the Dependencies graph DG
7.       Use KB(F) and DG to detect insider threat
8.       If an insider threat is detected
9.          Deny T
10.      Else
11.         Execute T
12.         Update KB(F) in the AZ J
13.   Else
14.      Send a "search request" for KB(F) to other availability zones
15.      Retrieve KB(F) from its host AZ, say HAZ
16.      Use KB(F) and DG to detect insider threat
17.      If an insider threat is detected
18.         Deny T
19.      Else
20.         Execute T
21.         Sends the Updated  KB(F) to HAZ
```

Mobile-Knowledgebases model eliminates the need to store knowledgebases of all insiders in every availability zone as in Peer-to-Peer model. Moreover, it has less traffic than model since updates of knowledgebases are sent to host availability zones only in case of "*moving*" insiders. Thus, this model is more scalable than Peer-to-Peer model. Furthermore, a failure of an availability zone does not affect the tasks of other insiders in other availability zones, which means it does not have a bottleneck as in the Centralized model (more reliable). Furthermore, in

most cases, the model needs to process the transactions and manage the knowledgebases of some insiders only, which means it has less processing overhead than other models.

This model can be optimized more in order to eliminate the need to send messages to all availability zones searching for the knowledgebase of a "moving insider". This can be achieved by storing a directory for all insiders on an organization and their hosting availability zones. Thus, when an insider's request is sent to an availability zone other than his/her hosting one, the cloud system at the new availability zone looks up the directory it has to retrieve the insider's hosting availability zone. Then, a message is sent to the hosting availability zone only to retrieve the knowledgebase of the insider. Storing the directory of all insiders needs more storage, but it greatly reduces the traffic overhead, especially when the number of availability zones and data centers gets larger.

7.4 Managing Dependency Graphs and Updates on Data items in Cloud RDBMS

Knowledgebases and dependency graphs are major parts in insider threat prediction and prevention models as discussed earlier. We suggest using the proposed dependency graphs, which are NDIG and CDG, in insider threat mitigation models in cloud RDBMS. In traditional insider threat mitigation models (in traditinoal RDBMS), dependency graphs are stored locally as a part of the models. In cloud RDBMS, the location of NDIG and CDG depends on which model we would adopt to manage knowledgebases. In Peer-to-Peer and Mobile-Knowledgebases models, dependecy graphs should be stored in each availability zone since insider threat prediction and prevention is performed at each one. However, in Centeralized model, dependency graphs need to be stored at the coordinater's sites only.

As discussed earlier, checking life times of data items in knowledgebases is crucial. Knowledgebase in cloud RDBMS should be managed by taking into account the lifetimes of data items such that expired data items are marked or deleted. Managing the lifetimes of data items in cloud RDBMS depends on the model used for managing knowledgebases. Two possible ways can be used to manage the lifetimes of data items in cloud relational databases (cloud RDB), which are *Exhaustive-Updating* and *Updating-on-Use*.

7.4.1 Exhaustive-Updating Approach

In this approach, on each write access of a data item by an insider, all knowledgebases of insiders are investigated searching for the data item. If the data item exists in one of to knowledgebases, the value of the data item is checked against expiration. If the value is expired, the data item is deleted or marked as expired. After completing this process, all instances of affected knowledgebases should be updated. Notice that in this approach a threat prediction model needs to investigate knowledgebase only to search for a risky data item and to check whether its lifetime is expired or not.

Using this approach in Peer-to-Peer model is time consuming, and poses network traffic and processing overhead since the Peer-to-Peer model maintains knowledgebases at each availability zone. Once a knowledgebase is updated, the updates should be sent through networks to other cloud RDBMS nodes. Therefore, this approach can be used in small systems that have small number of insiders and data items and when the number of cloud RDB instances is small. In Centralized Model, updating knowledgebases when using Exhaustive-Updating approach is performed on the coordinators site only. Moreover, network traffic occurs between the primary

site and the secondary site only. Thus, less network traffic is posed in comparing to the previous model, which means that it is more scalable. However, since all the processing of threat prediction and prevention is performed at coordinators site, using this approach adds more load to the coordinator's node, which may overload it. Therefore, powerful capabilities should be guaranteed and maintained at the node.

The workload of using this approach in Mobile- Knowledgebases model is distributed among availability zones. Clearly, since the knowledgebases of a group of insiders are stored in the closest availability zone, updating a knowledgebase is performed locally, and no update is sent out through networks. That is, no network overhead is posed as in Peer-to-Peer model, and contrarily to the Centralized Model, the processing overhead of maintaining knowledgebases is distributed among all availability zones. Thus, the best performance of the Exhaustive-Updating approach is achieved when it used with Mobile-Knowledgebases model.

7.4.2 Updating-on-Use Approach:

Contrary to the Exhaustive-Updating approach, this method does not update knowledgebases immediately after each update. Instead, the lifetime of a data item is updated when it is checked against insider threat only, which is performed as follows. At each read access to a data item by an insider, if the data item can be used with a another data item, say *F*, in the insider's knowledgebase to pose threat, the timestamp of *F* (in the insider's knowledgebase) is compared to the write timestamp of *F* in the cloud RDB. If *F* was updated after the last access to it by the insider, *F* is called *P-Expired*, which indicates *Possibly Expired*. Next, the value of *F* is investigated to check whether *F* is expired or not. If it is expired, the data item is removed from

the knowledgebase of the insider or marked as expired. This two-phase checking process eliminates the need to check the value of *F* in case it was not overwritten after the last access by the insider. Obviously, this approach reduces the processing overhead needed to investigate the entire knowledgebase at each write access. However, it adds more processing time to transactions since it needs checking both knowledgebases and cloud RDBMS in order to check the lifetimes of data items during transactions processing.

Adopting this approach in Peer-to-Peer model does not pose high network traffic overhead since updates to knowledgebases are sent gradually, which greatly less than the overhead that is posed when using Exhaustive-Updating approach. However, the processing time needed for transactions is greater than that needed in Exhaustive-Updating approach as mentioned earlier. Similarly, using this approach in Centralized Model reduces network traffic between the primary and secondary coordinators when compared to the Exhaustive-Updating approach, and adds more processing time to transactions. In Mobile-Knowledgebases model, no extra network traffic is posed as in Exhaustive-Updating approach. However, similar to the other models, more processing time is needed for transactions when adopting this approach.

8. CONCLUSIONS AND FUTURE WORK

8.1 Conclusions

The dissertation has studied insider threat in relational database systems. It has analyzed the factors and capabilities that insiders have and use to launch attacks. These factors include the accumulative knowledge that insiders get about data items and the dependencies and constraints that they acquire through legal accesses to data items or through collaborating with other insiders. The dissertation has classified dependencies into different types, and determined the amount and type of knowledge that insiders get based on dependencies.

Modeling the dependencies and constraints among data items facilitates understanding the structure of relational database systems. Furthermore, it enables defense systems to predict what knowledge an insider can get when accessing a data item. Therefore, we have developed two types of dependency graphs, which are Neural and Dependency Graph (NDIG) and Constraints and Dependencies Graph (CDG). NDIG demonstrates the dependencies among data items. Moreover, it shows the amount of information an insider can get about a data item when s/he accesses another data item or a group of data items. In comparison, CDG shows both the dependencies and constraints among data items. It tells what values of data items are stored in insiders' knowledgebases. In addition to dependencies and constraints, the dissertation has shown how knowledgebases play a major role in posing insider threat. To represent knowledgebases, the dissertation developed new knowledge graphs (KGs) that show the data items that insiders have accessed as well as the amount of information they have about data items. In addition, knowledge graphs demonstrate the data items about which insiders can infer

information. We have used knowledge graphs and dependency graphs to predict what knowledge (authorized or unauthorized) an insider may get if s/he accesses a data item. We have used a new graph called Threat Prediction Graph (TPG) that uses NDIG, CDG and KG to predict and prevent unauthorized knowledge. A threat alert is raised when an insider gets more information than allowed (based on a threshold value) about a data item. The dissertation has stated algorithms, theorems, proofs and simulations to prove the effectiveness of the proposed models in preventing unauthorized knowledge acquisition by insiders. The simulations have shown the effectiveness of the proposed models in preventing unauthorized knowledge without affecting the availability of data items. As shown by simulations, the percentage of prevented threat (without denying read accesses) increases as the number of transactions and the percentage of write operations in transactions increase, and it reaches about 30% when the percentage of write operations in transactions is 0.80 (when the number of transactions is 250).

Unauthorized modifications of sensitive data items are another aspect of insider threat. The dissertation has investigated this problem and addressed the paths insiders use to launch such attacks. It has developed new graphs called Modification Graphs and Dependency Graphs that show what data items an insider can change using legal write accesses or dependencies, and how to predict insider threat. Furthermore, we have proposed approaches to prevent such threats by hiding sensitive dependencies or denying some write access requests. In addition, we have stated the conditions under which those solutions are best used. That is, we have shown that in some cases allowing unauthorized modifications is better than hiding sensitive dependencies or denying write requests, especially when the cost of affecting availability is greater than the cost of exposing some sensitive information. Moreover, the dissertation has demonstrated when to

allow unauthorized modifications and when to prevent them based on the weight of sensitivity and availability values of data items. Algorithms, theorems, proofs and simulations have been provided to show the correctness and the effectiveness of the proposed approaches. The simulation has shown that the proposed approaches work better when the percentage of write operations in transactions gets larger.

The dissertation has addressed the importance of organizing accesses to data items in concurrent tasks. It has shown how organizing accesses to data items can prevent insider threats without affecting the availability of data items. We have shown how to compute the risk value of each possible sequence of executing tasks operations. Based on the risk values, a safe sequence is chosen and the operations of tasks are executed in the selected order. The dissertation has shown how to select a safe sequence in both declared and undeclared tasks. However, if no safe sequences are found, risky transactions are rejected. The effectiveness of the proposed approaches was tested using simulations. In declared tasks, the simulations have shown the percentage of prevented threat using safe sequences in comparison to all prevented threats. As reported by simulations, the probability of finding a safe sequence increases as the number of concurrent insiders and the percentage of write operations in transactions gets larger. The simulations have addressed that the percentage of prevented threats using safe sequences reaches about 65% when the number of concurrent insiders is 25 and the percentage of write operations equals to 50%. Moreover, the percentage of prevented threats using safe sequences reaches about 40% when the percentage of operations is 80% and the number of concurrent insiders is 10.

Security concerns are the major issues when moving data to the cloud. One of these concerns is the vulnerabilities that may be exploited by insiders to launch attacks. The dissertation has discussed insider threat in cloud relational databases. It has shown how balancing workload across availability zones and data centers may enable insiders to breach traditional insider threat prevention models. To prevent such threats, the dissertation has demonstrated new insider threat prediction and prevention models that are suitable for the cloud environment, which are *Peer-to-Peer* model, *Centralized* model and *Mobile-Knowledgebases* model. It has shown how knowledgebases, updates on data items and dependency graphs can be managed, synchronized and used effectively to defend cloud RDBMS against insider threat. Furthermore, it has addressed the conditions under which the models can work with highest performance, and has presented the advantages and disadvantages of each model on processing time, network traffic and overall cloud RDBMS performance.

8.2 Future Work

Defending cloud RDBMS against insider threat still needs more research. We plan to conduct research on organizing the operations of concurrent tasks in the cloud environment similar to what we have done for traditional relational databases. Moreover, we plan to conduct experiments to establish the effectiveness of the proposed models in managing knowledgebases in cloud RDBMS, and measure and compare the overhead (processing time and network traffic) that the models can add to the cloud RDBMS.

REFERENCES

[AES] J. Daemen and V. Rijmen. "AES Proposal: Rijndael, AES Algorithm Submission". Available Online at <http://www.nist.gov/CryptoToolkit>.

[Althebyan08a] Q. Althebyan and B. Panda. "A knowledge-base model for insider threat prediction". In Proceedings the IEEE Workshop on Information Assurance and Security.West Point, NY, USA, 2008.

[Althebyan08b] Q. Althebyan and B. Panda. "Performance analysis of an insider threat mitigation model". In Proceeding of the third IEEE International Conference on Digital Information Management. 2008.

[Amzaon12a] Amazon Elastic Compute Cloud. Available Online at <http://www.aws.amazon.com /ec2/>.

[Amazon12b] "Amazon Web Services: Overview of Security Processes". Available Online at <http://aws.amazon.com/security/>.

[Armbrust09] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. "Above the Clouds: A Berkeley View of Cloud Computing". Technical Report. University of California at Berkeley, 2009.

[Arshad09] J. Arshad, P. Townend and J. Xu,"Quantification of Security for Compute Intensive Workloads in Clouds: An automatic intrusion diagnosis approach for clouds,"International Journal of Automation and Computing,vol. 8, pp. 286-296, 2011.

[Azure12] SQL Azure. Available Onlilne at <http://www.microsoft. com/application platform/en/us/Key-Technologies/SQL-Azure.aspx>.

[Baraani96] A. Baraani-Dastjerdi, J. Pieprzyk, R. Safavi-Naini. "Security In Databases: A survey Study". University of Wollongong. 1996.

[Bertino09] E. Bertino, F. Paci, R. Ferrini, and N. Shang. "Privacy-preserving Digital Identity Management for Cloud Computing". IEEE Data Eng. Bull., pp.21-27. 2009.

[Bishop03] M. Bishop. "Computer Security: Art and Science". 1st Edition, Addison-Wesley, 2003.

[Bishop08] M. Bishop and C. Gates. "Defining the Insider Threat". In Proceedings of the 4th annual workshop on Cyber security and information intelligence research, Oak Ridge, Tennessee, USA. 2008.

[Biskup07] J. Biskup , D. Embley and J. Lochner. “Reducing inference control to access control for normalized database schemas”. Information Processing Letters, 2008.

[Boss] G. Boss, P. Malladi, D. Quan, L. Legregni and H. Hall. “Cloud computing”. Available Online at <www.ibm.com/developerworks/websphere/zones/ hipods>

[Brackney04] R. Brackney and R. Anderson. “Understanding the insider threat”. In Proceedings of 2004 workshop, Technical report, RAND Corporation, Santa Monica, CA,USA. 2004.

[Bradford05] 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, USA. 2005.

[Brodsky00] A. Brodsky, C. Farkas and S. Jajodia. “Secure Databases: Constraints, Inference Channels and Monitoring Disclosures”. IEEE Transactions on Knowledge and Data Engineering. 2000.

[CERT11] “The 2011 CyberSecurity Watch Survey”. Available Online at <www.cert.org/>

[Chagarlamudi09] M. Chagarlamudi, B. Panda, and Y. Hu. “Insider Threat in Database Systems: Preventing Malicious Users' Activities in Databases”. In Proceedings of the 6thInternational Conference on Information Technology, Las Vegas, NV, USA, pp.1616-1620. 2009.

[Chinchani05] R. Chinchani, A. Iyer, H. Ngo and S. Upadhyaya. “Towards a Theory of Insider Threat Assessment”. In Proceedings of the 2005 International Conference of Dependable Systems and Networks (DSN’05). 2005.

[Chow09] R. Chow , P. Golle , M. Jakobsson , E. Shi , J. Staddon ,R. Masuoka and J. Molina. “Controlling data in the cloud: outsourcing computation without outsourcing control”. In Proceedings of the 2009 ACM workshop on Cloud computing security. 2009.

[Chung99] C. Chung, M. Gertz and K. Levitt. “DEMIDS: A Misuse Detection System for Database Systems”. In Proceedings of the IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Controlling Information Systems, Kluwer Academic Publishers. 1999.

[Conway72] R. Conway, W. Maxwell and H. Morgan. “On the Implementation of Security Measures in Information Systems”. Communications of the ACM, 1972.

[Cooper08] R. Cooper. "Verizon Business Data Breach Security Blog". Available Online at <www.securityblog.verizonbusiness.com/2008/>

[Curnio] C. Curino, E.P.C. Jones, R.A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan, and N. Zeldovich. "Relational Cloud: a Database Service for the cloud". In Proceedings of CIDR. 2011

[Das11] S. Das, S. Nishimura, D. Agrawal, and A.E. Abbadi. "Albatross: Lightweight Elasticity in Shared Storage Databases for the Cloud using Live Data Migration". In Proceedings of the 37th International Conference on Very Large Data Bases (VLDB). 2011.

[David04] D. Woodruff and J. Staddon. "Private inference control". In Proceedings of the 11th ACM conference on Computer and communications security. 2004.

[Dawson99] S. Dawson, S.De Capitani di Vimercati, and P. Samarati. "Specification and enforcement of classification and inference constraints". In Proceedings of the 20th IEEE Symposium on Security and Privacy. 1999.

[DES] National Bureau of Standards. "DES modes of operation". Available Online at <csrc.nist.gov>

[Elmore11] A. Elmore, S. Das, D. Agrawal and A.E. Abbadi. "Zephyr: live migration in shared nothing databases for elastic cloud platforms". In Proceedings of the SIGMOD Conference. 2011.

[Farkas01] C. Farkas, T. Toland and 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. 2001.

[Farkas07] C. Farkas and S. Jajodia. "The Inference Problem: A Survey". ACM SIGKDD Explorations. 2007.

[Forrester11] "The Value of Corporate Secrets". Available Online at <www.microsoft.com>

[Gordon05] L. Gordon, M. Loeb, W. Lucyshyn and R. Richardson. "Computer Crime and Security Survey". Available Online at <http://www.cpppe.umd.edu/>

[Garfinkel02] R. Garfinkel, R. Gopal, and P. Goes. "Privacy Protection of Binary Confidential Data against Deterministic, Stochastic, and Insider Threat". Management Science. 2002.

[Hacigumus12] H. Hacıgümüş, J. Tatemura, Y. Chi, W. Hsiung, H. Jafarpour, H. Moon and O. Po. "CloudDB: A Data Store for All Sizes in the Cloud". Available Online at <http://www.nec-labs.com/dm/CloudDBweb.Pdf>.

[Harrison76] M. Harrison , W. Ruzzo and J. Ullman. "Protection in operating systems". Communications of the ACM. 1976.

[Heping05] Y. Heping, L. Bing, Y. Xiaoming, W. Wei, S. Baile and Y. Genxing, "Controlling FD and MVD inferences in MLS". In Proceedings of the 5th International Conference on Computer and Information Technology. 2005.

[Hu03] Y. Hu and B. Panda. "Identification of Malicious Transactions in Database Systems". In Proceedings of the 7th International Database Engineering and Applications Symposium. 2003.

[Hu06] N. Hu, P. Bradford and J. Liu. "Applying Role Based Access Control and Genetic Algorithms to Insider Threat Detection". In Proceedings of the 44th ACM Southeast Conference. 2006.

[Hwang09] K. Hwang, S. Kulkarni, and Y. Hu. "Cloud Security with Virtualized Defense and Reputation-Based Trust Management". IEEE International Conference of Dependable, Autonomic, and Secure Computing (DASC 09), IEEE CS Press. 2009.

[Jabbour09a] G. Jabbour and D.A. Menasc´e. "Stopping the Insider Threat: the case for implementing autonomic defense mechanisms in computing systems". In Proceedings of the International Conference of Information Security and Privacy. 2009.

[Jabbour09b] G. Jabbour and D.A. Menascé. "The Insider Threat Security Architecture: A Framework for an Integrated, Inseparable, and Uninterrupted Self-Protection Mechanism". In Proceedings of the international Conference on Computational Science and Engineering. 2009.

[Kaufman09] L. Kaufman. "Data Security in the World of Cloud Computing". IEEE Security & Privacy. 2009.

[Lampson71] B. Lampson. "Protection". In Proceedings of the 5th Princeton Conference on Information Sciences and Systems. 1971.

[LII] "Information Security". Available Online at <http://www.law.cornell.edu/uscode/ 44/35 42.html>

[Marks96a] D. Marks. "Inference in MLS Database Systems". IEEE Transactions on Knowledge and Data Engineering. 1996.

[Marks96b] D. Marks , Amihai Motro and Sushil Jajodia. "Enhancing the Controlled Disclosure of Sensitive Information". In Proceedings of the 4th European Symposium on Research in Computer Security. 1996.

[Mathew10] S. Mathew, M. Petropoulos, H. Ngo and S. Upadhyaya. "A Data-Centric Approach to Insider Attack Detection in Database Systems". In Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection. 2009.

[Maybury05] 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. 2005.

[Meza05] B. Aleman-Meza, P. Burns, M. Eavenson, D. Palaniswami, A. Sheth. "An Ontological Approach to the Document Access Problem of Insider Threat". In Proceedings of the International Conference on Intellegence and Security Informatics, 2005.

[Morgenstern87] M. Morgenstern. "Security and Inference in Multilevel Database and Knowledge-Base Systems". ACM SIGMOD Record. New York, Vol. 16, pp. 357 – 373, December 1987.

[Motahari09] S. Motahari, S. Ziavras, M. Naaman, M. Ismail and Q. Jones. "Social Inference Risk Modeling in Mobile and Social Applications". In Proceedings of CSE (3). 2009.

[Murata89] T. Murata. "Petri nets: Properties, analysis and applications". In Proceedings of the IEEE. 1989.

[NIST95] "An Introduction to Computer Security: The NIST Handbook". Available Online at <http://www.nist.gov>.

[Park06] S. Park, and J. Giordano. "Role-based Profile analysis for scalable and accurate insider-anomaly detection". In Proceedings of the 25th IEEE International Performance, Computing and Communications Conference. 2006.

[Probst10] C. Probst, J. Hunker, D. Gollmann and M. Bishop (Eds.). "Insider Threat in Cyber Security". 1st Edition, Springer. 2010.

[RDS12] Amazon Relational Databases. Available Online at <http://aws.amazon.com/rds/>

[Rivest78] R. Rivest, A. Shamir, L. Adleman. "A method for obtaining digital signatures and public-key crypto systems". Communications of the ACM, 1978.

[Shanon48] C. E. Shannon. "A mathematical theory of communication". Bell System Technical Journal, (1948).

[Spitzner03] L. Spitzner. "Honeypots: Catching the Insider Threat". In Proceedings of the 19th Annual Computer Security Applications Conference. 2003.

[Stachour90] P. Stachour , B. Thuraisingham. "Design of LDV: A Multilevel Secure Relational Database Management". IEEE Transactions on Knowledge and Data Engineering. 1990.

[Stryer12] P. Stryer. "Understanding Data Centers and Cloud Computing". White Paper, Available Online at <http://www.globalknowledge.se/pdf/ WP_DC_ DataCenterCloudComputing1.pdf>.

[Subashini10] S. Subashini and V. Kavitha. "A Survey on Security Issues in Service Delivery Models of Cloud Computing". Journal of Network and Computer Applications. 2010.

[Su87] T. Su and G. Ozsoyglu. "Data dependencies and inference control in multilevel relational database systems". In Proceedings of the IEEE Symposium on Security and Privacy. 1987.

[Su91] T. Su and G. Ozsoyoglu. "Controlling FD and MVD Inferences in Multilevel Relational Database Systems, IEEE Transactions on Knowledge and Data Engineering. 1991.

[Takabi10] H. Takabi, J.B.D. Joshi, and G. Ahn. "Security and Privacy Challenges in Cloud Computing Environments". IEEE Security & Privacy. 2010.

[Wang09] C. Wang, Q. Wang, K.Ren, W. Lou. "Ensuring Data Storage Security in Cloud Computing". In Proceedings of 17th IEEE International Workshop on Quality of Service. 2009.

[White09a] J. White, B. Panda, Q. Yaseen, W. Li and K. Nguyen. "Detecting Malicious Insider Threats using a Null Affinity Temporal Three Dimensional Matrix Relation". In Proceedings of the 7th International Workshop on Security in Information Systems. 2009.

[White09b] J. White and B. Panda. "Automatic Identification of Critical Data Items in a Database to Mitigate the Effects of Malicious Insiders". In Proceedings of the the 5th International Conference on Information Systems Security. 2009

[Yalamanchili04] R. Yalamanchili and B. Panda. "Transaction Fusion: A Model for Data Recovery from Information Attacks". Journal of Intelligent Information Systems. 2004.

[Yaseen09] Q. Yaseen and B. Panda. "Knowledge Acquisition and Insider Threat Prediction in Relational Database Systems". In Proceedings of the International Workshop on Software Security Processes. 2009.

[Yaseen10a] Q. Yaseen and B. Panda. "Organizing Access Privileges: Maximizing the Availability and Mitigating the Threat of Insiders Knowledgebase". In Proceedings of the 4th International Conference on Network and System Security. 2010.

[Yaseen10b] Q. Yaseen and B. Panda. "Predicting and Preventing Insider Threat in Relational Database Systems". In Proceedings of the 4th IFIP WG 11.2 Workshop in Information Security Theory and Practice. 2010.

[Yaseen10c] Q. Yaseen and B. Panda. "Malicious Modification Attacks by Insiders in Relational Databases: Prediction and Prevention". In Proceedings of the Second IEEE International Conference on Information Privacy, Security, Risk and Trust (PASSAT2010). 2010.

[Yaseen11] Q. Yaseen and B. Panda. "Enhanced Insider Threat Detection Model that Increases Data Availability". In Proceedings of the 7th International Conference of Distributed Computing and Internet Technology. 2011.

[Yaseen12a] Q. Yaseen and B. Panda, "Mitigating Insider Threat without Limiting the Availability in Undeclared Tasks", In Proceedings of the 6th IEEE Conference on Software Security and Reliability (SERE2012), Washington D.C, June 2012 [To Appear].

[Yaseen12b] Q. Yaseen and B. Panda, "Insider Threat Mitigation: Preventing Unauthorized Knowledge Acquisition", International Journal of Information Security, 2012 [To Appear].

[Yip98] R. Yip and K. Levitt. "Data Level Inference Detection in Database Systems". In Proceedings of the 11th IEEE Computer Security Foundations Workshop. 1998.

[Zuo04] Y. Zuo and B. Panda. "Fuzzy Dependency and its Applications in Damage Assessment and Recovery". In Proceedings of the 5th Annual IEEE Information Assurance Workshop. 2004.