Predicting and Preventing Insider Threat in Relational Database Systems

Qussai Yaseen and Brajendra Panda

Dept. of Computer Science and Computer Engineering,
University of Arkansas, Fayetteville, AR, USA.
{qyaseen,bpanda}@uark.edu

Abstract. This paper investigates the problem of insider threat in relational database systems. It defines various types of dependencies as well as constraints on dependencies that may be used by insiders to infer unauthorized information. Furthermore, it introduces the Constraint and Dependency Graph (CDG), and the Dependency Matrix that are used to represent dependencies and constraints on them. Furthermore, it presents an algorithm for constructing insiders knowledge graph, which shows the knowledgebase of insiders. In addition, the paper introduces the Threat Prediction Graph (TPG) to predict and prevent insider threat.

Keywords: Relational Database, Security, Insider Threat, Dependencies.

1 Introduction

Recently, many researchers have dealt with the problem of insider threat. It is defined as the threat that is caused by a malicious insider who has authorized access privileges and knowledge of the computer systems of an organization, and is inspired to antagonistically influence the organization [1]. It is considered as important as the problem of the outsider threat (hackers), due to the extreme harm that may cause. Trusted insiders are responsible of 52% of all security breaches in 2004, according the 2005 FBI Computer Crime Survey [2].

Obviously, insider threat is a very critical issue. Many mechanisms have been proposed for protecting data from outside attacks. However, those mechanisms fail to protect data from authorized users who may misuse their privileges to make harm to systems. Thus, finding mechanisms that protect sensitive data against insiders has become a key demand due to the amount of harm that can be caused by those malicious insiders.

To the best of our knowledge, except what is presented in [3] and [4], no other research has been performed exclusively on insider threat in relational databases. In order to clearly understand the problem with insider threat in relational database systems, this paper explains various types of dependencies between relational database data items that can be used by insiders to launch their attacks. Moreover, it gives a clear view of the problem at various granularity levels, that is, at the table, record, and attribute levels. It describes the conditions

or constraints that should be satisfied in order to make changes to dependent data items. Then, a new type of dependency graph, called the Constraint and Dependency Graph CDG, and the concept of dependency matrix, which are used to represent the different types of dependencies and constraints have been explained. These data structures are used to build the knowledge graphs of insiders. Following that, the Threat Prediction Graph TPG, which is used for insider threat prediction and prevention, has been introduced.

The rest of the paper is organized as follows. Section 2 presents the previous work. Section 3 investigates various types of dependencies. Section 4 demonstrates the types of constraints on dependencies. The insider threat mitigation mechanism has been presented in section 5. Section 6 provides the conclusions and future work.

2 Related Work

Different definitions of insider threat have been introduced in [1] and [5]. Researchers in [3] have discussed this issue at an application level such as relational databases. In general, they agreed that the insider is the person who has access privileges to and is familiar with the system under consideration and is inspired to make harm to that system. In [6], the definition of insider has been extended to three classes.

As we stated before, so far very little work has been performed in this area. The research presented in [7] used existing methods of detecting external threat, such as using honeypots, to detect insider threat. Althebyan and Panda [8] introduced the knowledge graph of an insider at the system level as well as the dependency graph between data items. They used these graphs to predict and prevent insider threats. However, their work is at the system level and does not consider relational databases.

Insider threat in relational databases depends mainly on dependencies. Dependencies as well as the inference problem have been discussed extensively in [9–12]. Farkas et al. [9, 10] discussed how users can get sensitive data using non-sensitive data to which they have no access. Brodsky et al. [11] and Yip and Levitt [12] discussed the inference channels problem and addressed how to discover inference channels using queries.

To the best of authors knowledge, the work in [3] was the first work that introduced the insider threat in relational databases. In that work, a new dependency graph, called the neural dependency and inference graph, was defined, which shows the dependencies between data items as well as the amount of information an insider can acquire (legally or illegally) about different data items. This graph is used along with the knowledge graph to predict and prevent insider threat. In the following sections we discuss our model. We start with the description of dependency relationships among data items in a relational database system.

3 Various Types of Dependencies

Two data items A and B have a dependency relationship between them if one of them depends on the other or if they depend on each other. A dependency between A and B is represented by the notation $A \rightarrow B$, which means that B depends on A. A dependency relationship is classified according to a number of categories, such as the *strength*, *direction*, and the *transitivity*. The strength of a dependency relationship is classified into two types: weak and strong, which are defined as follows.

Definition 1 (strong dependency). Given the dependency $A \rightarrow B$, where A and B are two data items, if a change in A results in a change in B, then it is a strong dependency and is represented by $A \Longrightarrow B$.

Definition 2 (weak dependency). The dependency $A \rightarrow B$ is called weak, if a change in A may not result in a change in B. Such a dependency is represented by A $\Rightarrow$ B.

The direction indicates the source and the destination of a dependency, which is classified into one_way and two_way (cyclic) dependency. The following two definitions explain these types.

Definition 3 (one_way dependency). Given the dependency $A \rightarrow B$, if B depends on A but A does not depend on B, then this dependency is called one_way and is represented by $A \rightharpoonup B$.

Definition 4 (cyclic dependency). If for the dependency $A \rightarrow B$, B depends on A and A depends on B, then it is called a cyclic dependency and is represented by $A \rightleftharpoons B$.

A cyclic dependency contains two relationships between the two data items, which may be both strong, both weak, or one of them is weak and the other is strong. Based on the transitivity property, we classify dependency relationships into direct and transitive dependencies as defined below.

Definition 5 (direct dependency). Given the dependency $A \rightarrow B$, where A and B are two data items, then the dependency is called direct dependency if a change in A directly affects (make a change in) B and represented by $A \mapsto B$.

Definition 6 (transitive dependency). Consider the three data items A, B, and C, where C depends on B and B depends on A. If a change in A makes a change in B and this change in B makes a change in C, then C depends transitively on A and the relation is represented by $A \rightsquigarrow C$.

These dependencies may be used by insiders to infer unauthorized information. For instance, assume an insider has read access on a data item B, which depends strongly on another data item A. Then, a change in B makes the insider infer that a change has been occurred to A. Moreover, if the insider knows the constraints on the dependency (constraints on dependencies are discussed

in the next section), he/she can deduce the new value of A. Equivalently, if the value of B is not changed, then the insider knows that the value of A has not been changed. However, if B depends weakly on A, then if the value of B is not changed, this does not mean that the value of A is still the same. Likewise, direct dependency and transitive dependency determines whether an insider may infer information directly or transitively.

Relational databases have different levels of granularities. We consider them as the low level (attribute level), the intermediate level (record level) and the high level (table level). In this paper, we discuss these types of dependencies at various levels of granularities. All types of dependencies, except the cyclic dependency, are found at the attribute level. Such examples are easy to find. Actually, some relational database systems may have a cyclic dependency at the attribute level. Similarly, all types of dependencies are observed at the table level since a table inherits the dependencies present at its attribute levels, that is, a dependency between two tables is basically a dependency between attributes that belong to them. However, two tables may have more than one type of dependencies.

Similarly, records inherit dependencies from their attributes. This means that various types of dependencies exist at record level also. Two records having a dependency relationship may exist in the same table or in two different tables. When the records belong to the same table, there are two possibilities: First, an attribute depends on itself, which is called a self-dependent attribute. For instance, consider the Rank attribute for a student table. Values of this attribute are dependent on each other. In other words, a change in a student's rank affects the rank of other students' records. Secondly, the attributes of the dependency are different. This case exists when a self dependent attribute is also dependent on another attribute. For example, consider the dependency GPA (Rank in a student table. A change in the GPA attribute for a student may make a change in the Rank attribute of the student, which, as a result, makes a change in the Rank attribute of other records (student). In other words, a change in the GPA field of a record may make a change in the Rank field of another record.

4 Constraints on Dependencies

In the previous section we discussed various types of dependencies at different granularity levels. In some cases, a dependency relationship may involve a constraint; that is, a change on the dependent data item occurs only when the specified constraint is satisfied. We classify such constraints into two types: changing the value of an attribute and deleting or inserting records. Next, we discuss how to represent the constraints of the first type where as the second type is discussed section 4.2.

4.1 Using Petri Nets for Representing Constraints and Dependencies

We have developed a method to represent the dependencies and constraints in order to investigate the flow of information between different data items at

different granularities. Some constraints could be complex. In some cases, a dependency relationship can be represented using a formula where as in other cases it is difficult to represent dependencies that way. For instance, consider the dependency constraints between the two attributes a_1 and a_2, that says the value of a_2 must equal c_3 when a_1 is in the range c_1 and c_2. And a_2 must equal $c_4$when the value of a_1 is less than c_1; otherwise, the value of a_2 is c_5, where c_1, c_2, c_3, c_4 and c_5 are constants. We use Petri Nets [13] to construct a dependency graph that represents all types of dependencies as well as constraints. We call such a graph as the Constraint and Dependency Graph (CDG).

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 the weight of the corresponding arc. The properties of Petri Nets and other details are beyond the scope of this paper. Interested readers may refer to [13]. Figure 1 shows an example Petri Net representing the formula $X = 2\,a + 3\,b$. In the rest of this paper, the conditions of firing are assumed to be satisfied always, so the tokens are not shown.

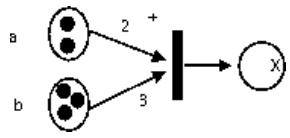

Fig. 1. Using Petri Nets to Represent a Formula.

To show further examples of how Petri Nets can be used to represent dependencies and constraints between data items in relational databases, consider two tables T_1(a_1,a_2,a_3) and T_2(a_4,a_5,a_6,a_7) having the following dependencies and constraints:

- $c_1 \leq a_1 \leq c_2 (a_2 = c_3$
- $c_1 > a_1 (a_2 = c_4$
- $a_1 > c_2 (a_2 = c_5$
- $a_4 = 3\,a_3 + 1$
- $a_6 = 2\,a_2 + 3\,a_5$

In addition, suppose that the attribute a_2 is a self dependent attribute. Figure 2 shows the CDG of this relational database using Petri Nets. In the figure, one of the three transitions connected to the attribute a_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 a_2. This makes the attribute a_2 equals the value of the constant from which the token comes. This represents the dependency and constraints between a_1 and a_2. Notice that each attribute name is preceded by the name of the table to which it belongs. The self loop on the attribute a_2, which is represented by an arrow from the place of the attribute a_2 to a transition and an arrow from the transition to the same place, means that it is a self dependent attribute. The dependency between the attributes $\{a_2, a_5\}$ and the attribute a_6 is represented using the same way of representing the formula in Figure 1. Similarly, the dependency between a_3 and a_4 is represented using the same way. The transition with no inputs is called a source transition. It is used in this graph to make three copies of the attribute a_1.The purpose of doing this is to explain that firing a_6 does not always depend on firing a_1. That is, an indirect change in a_6 may be caused by a change in a_1 or a change in a_2.

CDGs are used to build knowledge graphs of insiders. They are used to show how insiders can follow dependencies to infer knowledge about data items to which they do not have privileges. Constraints in the CDG show what values of data items are stored in the knowledgebase of insiders exactly. Details of how dependencies in the CDG are used to construct such graphs are discussed later in Section 5.

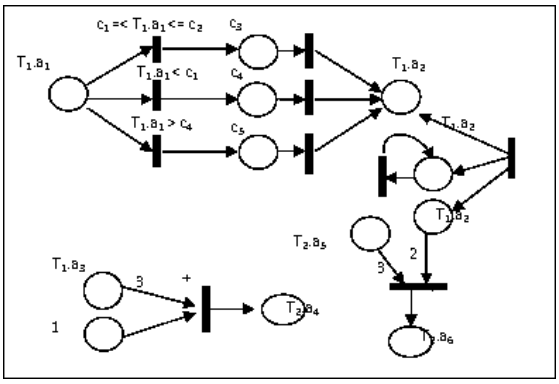

Fig. 2. A Constraint and Dependency Graph CDG.

The above figure shows how Petri Nets can be used efficiently to construct a dependency graph to represent the dependencies and constraints. Moreover, since each attribute name is preceded by the corresponding table name, both attribute and table levels 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. However, a dependency between records exists in the same table when there is a self dependent attribute in that table. In this case, the self dependent attribute is represented by a loop on the attribute (such as the loop on the attribute a_2) to show such dependency.

Insiders may be able to predict values of data items, which they may not be authorized to access, by investigating constraints on dependencies. For instance, in Figure 2, assume that an insider has read access on attribute a_2 and has no access to attribute a_1. Then, if the value of a_2 is changed to the value c_3, the insider, who knows the constraint, would realize that the value of a_1 is changed to the c_1. Similarly, the insider can deduce the correct value of a_1 if the value of a_2 is changed to either c_4 or c_5. Thus, insiders may use their knowledge about data dependencies and constraints among them to acquire knowledge about some data items that are inaccessible to them.

4.2 The Dependency Matrix

The type of constraint that is discussed earlier is to change a value of some attribute to a specific value. However, besides this, the record and table level have another type of constraint which is a deletion or an insertion of a record. That is, 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, 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 a record to the "Dependents" table) of an employee changes the value of the health insurance of the employee. Basically, this insertion or deletion changes the related record of the corresponding employee. On the other hand, insertion or deletion of a record in a table that has a self dependent attribute may affect other records in the same table. This type of constraint exists in both the table and the record levels.

To represent both types of constraints at the table level (and implicitly, the record level), a dependency matrix is used. Figure 3 represents an example of a dependency matrix that shows dependencies between different tables as well as the constraints on such dependencies. The x and y axis represent tables. Each cell contains a set of pairs (C, T), where C denotes a constraint and T denotes the type of the dependency, where 2 means a strong dependency and 1 means a weak dependency. For instance, the cell (T_1, T_2) means that if the constraint C_1 is satisfied on T_1, then a change on T_2 should 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.

Definition 7 (Safe Cluster). Given a dependency matrix of a relational database, a safe cluster $C = \{T_1, \ldots, T_n\}$ is a cluster of tables in which each table T_i

is independent ,directly and transitively, from all other tables that belong to the same cluster. Thus, a read access to any table in this cluster does not compromise information about other tables. In addition, a change to any table in this cluster does not affect other tables.

Similarly, a hot cluster is defined as follows.

Definition 8 (Hot Cluster). Given a dependency matrix of a relational database, a hot cluster $C = \{T_1, \ldots, T_n\}$ is a cluster of tables in which each table T_i is directly dependent on all other tables that belongs to the same cluster. That is, a read access to any table in this cluster compromises information about other tables. In addition, modifying some values in any table in this cluster affects other tables.

	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	-

Fig. 3. A Dependency Matrix

Depending on Figure 3, Figure 4 shows an example of hot and safe clusters. For instance, clusters 3 to 6 are safe clusters. Similarly, the clusters $C_1 = \{T_2, T_4\}$ and $C_2 = \{T_5, T_3\}$ are hot clusters since if the insider accesses one or more tables of them, he/she acquires information from other tables in the same cluster. In addition, he/she can make changes in them. As shown in figure 4, hot clusters and/or safe clusters may overlap. Notice that tables that belong to different clusters may still have a dependency relationship, but not cyclic. For instance, tables T_1 and T_2 still have one way dependency.

5 Insider Threat

Insiders may use their knowledge about dependencies and constraints to get unauthorized information. This section discusses this problem as well as the possible solutions.

5.1 Insider's Knowledgebase

This section investigates the knowledgebase of insiders. Knowledgebase determines to which data items an insider has authorized and unauthorized read access. It is constructed based on the different levels of granularities of relational

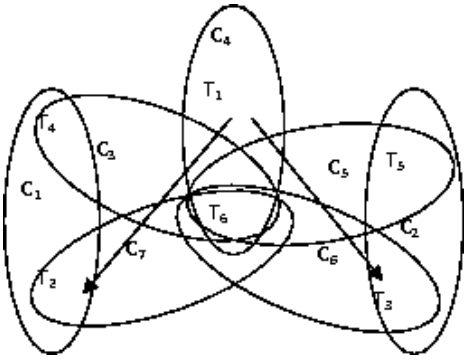

Fig. 4. Hot and safe clusters.

databases, which are the table, the record and the attribute levels. In light of the previous section, the following are concluded. First, if an insider gets read access to a table that belongs to a hot cluster, he/she can acquire information about all other tables in that cluster. Second, if an insider gets read access to a table in a safe cluster, he/she cannot infer any information about any other table that belongs to the same cluster. Finally, if an insider gets read access to a table in some cluster, he/she still can infer information about other dependent tables that belong to different clusters. This case occurs when the dependency between them is one way dependency.

In addition to the above conclusions, an insider can acquire information about other tables transitively, where the transitive inference is defined as follows:

Definition 9 (transitive inference). Given the dependency $T_1 \rightarrow T_2 \ldots T_{n-1} \rightarrow T_n$, where T_1 to T_n are tables in a relational database, the insider who has read access to T_1 can infer information about T_j, where $j = 3$ to n, which is called transitive inference, if the following two conditions are satisfied:

1. He/she knows the dependency and constraints between tables T_1 to T_j.
2. The dependencies between the tables T_1 to T_n are between data items in the form $C_1 \rightarrow C_2 \ldots C_{n-1} \rightarrow C_n$, where $C_j \in T_j$.

In addition, the insider who has access to T_n can infer information backward until T_1 by the same way.

However, tables in the knowledgebase of an insider do not necessarily mean that the insider can infer all information about those tables or can change whatever he/she wants in them. To reveal more details about this, dependencies and constraints between attributes in those tables should be investigated. To do so, the dependency graph CDG is used. For instance, suppose that an insider has full read and write access on table T_1 in Figure 2. Both tables T_1 and T_2 are in the knowledgebase of the insider since they have a dependency relationship. The insider knows all information about T_1, whereas his/her read access to 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, he/she infers information about a_4 and a_6 and acquires information about a_5 by cyclic inference [3], but he/she does not have information about other attributes in T_2. To compute how much information the insider has about specific attributes, NIDG in [3] is used.

A similar scenario is used in the case of records level. If the insider has read access to a record, then this record is in his/her knowledgebase. In addition, records that depend on this record are there also. But, this does not mean that he/she has full information about those records. To investigate deeply what information the insider has about those records, the dependency between attributes should be investigated.

5.2 Knowledgebase Algorithm

Algorithm 1 shows the algorithm for building the multilevel knowledge graph, which represents knowledgebase at different levels of granularity. The algorithm assumes that the insider is familiar with dependencies and constraints. It uses the NIDG as well as CDG of the relational database under consideration as well as the dependency matrix. In addition, it uses the hot and safe clusters to facilitate construction of the knowledge graph.

The algorithm adds the insider as a root of the knowledge graph. The second level of the graph contains the tables to which the insider has read access (directly or by inference). For each table in the second level, the algorithm determines to which attributes the insider has read access. The NIDG is used to label edges by the amount of information the insider can have about each data item (attribute or table). Either the NIDG or CDG is used to show dependencies between knowledge units (attributes), which are represented by an edge (arrow) from the source attribute to the destination (dependent) attribute. Moreover, the CDG is used to show what values of attributes are stored in the knowledgebase of the given insider as mentioned earlier, which is used in insider threat prediction and prevention later in section 5.4. Notice that the amount of information the insider has about a table is the average of all information he/she has about all attributes belonging to the table.

The following example clarifies how the algorithm works. Suppose that the corresponding NIDG for the CDG in Figure 2 is as shown in Figure 5. Let the insider has read access to T_1, then Figure 6 shows the knowledge graph of the insider. Dashed lines represent the paths the insider follows to acquire the information. For instance, he/she acquires information about the attribute a_2 using direct access, whereas he/she gets information about $a_6$by inference using the attribute a_2.

Weights on edges show the amount of information the insider has about the destination data items. The weights on edges between the root and the tables are based on the assumption that tables do not have attributes other than those shown.

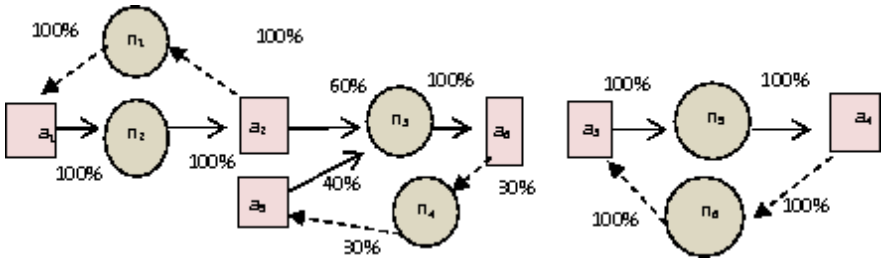

Fig. 5. The NDIG of the Database in Figure 2.

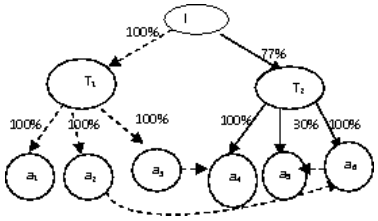

Fig. 6. Knowledge Graph KG of an insider.

5.3 Insider Threat Prediction and Prevention

Constructing the knowledge graph of an insider helps in predicting and preventing the insider threat (getting unauthorized information). To achieve this goal, the threat prediction graph TPG is used, which is built based on the knowledge graph. A threat prediction value TPV is stored in each attribute that belong to the knowledgebase of the insider, which is defined as follows.

$$\mathrm{TPV}(k) = f(k)/T(k) \tag{1}$$

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 (the amount of information that the insider is allowed to get about k) according to the insider under consideration. The TPG is defined as follows.

Definition 10. The Threat Prediction Graph TPG (V,E ,L) is a graph that is used to predict and prevent the threat that insiders pose, where:

1. V indicates nodes, such that:
   (a) The root represents the insider (root node).
   (b) The second level of nodes (with labels $T_i$inside) represents the tables about which the insider has knowledge (tables' nodes).
   (c) The nodes that are labeled as a_i are attributes (attributes' nodes).
2. E indicates the edges, such that:

(a) Dashed edges represent the paths that the insider follows to get knowledge about the destination attribute.
(b) Solid edges represent which table the destination attribute belongs to.

3. L represents the values inside the attribute nodes, where a value inside such node represents the threat prediction value of the corresponding attribute.

To construct the TPG, all data items to which the insider has read access are broken into knowledge units (attributes). Then, the NDIG, the KG and the set of threshold values according to the given insider are used to construct the TPG. A knowledge unit is considered a threat if its TPV is greater than one.

The TPG prevents insider threat as follows. Suppose an insider requested access to a knowledge unit, say RKU. If its TPV is greater than one, deny this request. Otherwise, add it to the TPG. Then, check if granting an access to this knowledge unit causes a threat. That is, check if it causes the TPV of another knowledge unit, say KU_j, to be greater than 1. In this case, administrators have two choices: First, do not grant access to the RKU. Second, if the insider needs RKU to perform a necessary job, administrators can grant the insider an access to RKU but revoke access(es) to some knowledge unit(s) that have the following properties. First, they already exist in the knowledgebase of the insider. Second, they can be used in conjunction with RKU to compromise unauthorized information about KU_j. Third, revoking access to them preserves the safety of all attributes. Finally, the lifetime of those knowledge units are expired.

The last point states that when the knowledge unit is expired, the insider infers a wrong value if he/she uses this value in inference. Farakas et al. [9] considers that a knowledge unit is expired if it has been changed after the last access to it by the insider. However, in this work, 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. For instance, assume that the following constraint exists.

$$(c_1 < a_1 < c_2) \wedge (c_3 < a_2 < c_4) \rightarrow a_3 = c_5 \ .$$

Suppose that the threshold values for an insider about a_1, a_2 and a_3 are 100%, 100% and 60% respectively. Also, assume that the insider got about 20% of knowledge about $a_3$by having direct access to a_1. There is no threat so far. Now, suppose that the value of a_1 is updated to c_6, where $(c_1 < c_6 < c_2)$, and then the insider requests access to a_2. Using the threat prediction graph, administrators can grant access to a_2 since the threshold is 100%. However, granting the access causes a threat since the TPV of a_3 will be 1.7. Now, administrators have two choices, one of them is to grant the insider an access to a_2 but to revoke the access to a_1 if it is expired. In this case, a_1 has been updated since the last access of the insider. If they consider that a_1 is expired, because it was updated, and choose this choice, the threat is not removed. This is clear since the insider still can use the old value of a_1 and the value of a_2 to compromise the exact

value of a_3. This means that the lifetime of a knowledge unit does not expire until its value is changed such that its old value cannot be used to make correct inference. For instance, if a_1 is changed to a_7, where either $a_7 > c_2$ or $a_7 < c_1$, then the value of a_1 is expired.

6 An Example Scenario

Suppose that figure 7 represents the NIDG of a relational database, where the attributes a_1 and a_5 are in the table T_1, a_2 is in the table T_2, and a_3, a_4 and a_6 are in the table T_3. Also, assume that the set of attributes to which the insider has direct access is $\{a_1\}$. Then, the knowledgebase of the insider is $\{(a_1, 100\%), (a_5, 11\%), (a_2, 12\%)\}$. Notice that the amount of knowledge about a_2 is acquired by cyclic inference. This value is a supposed value since it is not shown in the graph. Actually, weights on edges in the graph are the amount of information the insider gets if he/she has exact knowledge about source data items. Thus, in the example, the amount of information the insider gets about a_2 is not 19% since the insider does not have exact knowledge about a_5.

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 of data items according to the given insider. The initialized TPG for this insider is shown in figure 8 (a).

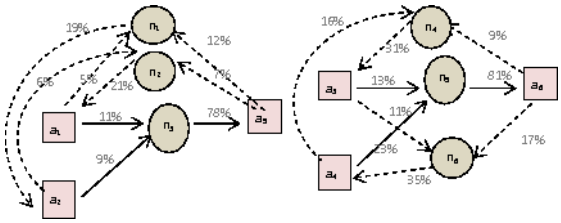

Fig. 7. A NIDG of a Relational Database.

Assume that the insider requested access to a_3. Obviously, granting access to a_3 does not form any threat as shown in figure 8 (b). 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, we assume that the TPV is 0.15. Now, suppose that the insider 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 8 (c). In this TPG, the TPV of a_6 is greater than 1, which indicates a threat. In this case, the administrator has two choices. First, the administrator does not grant the insider access to a_4. Second, the administrator grants the insider access to a_4, but revokes his/her access to a_3 (if its lifetime is expired). If the administrator chooses the second option, the TPG of the insider would look as shown in Figure 8 (d). Dashed arrows show paths that the insider follows to get

information about destination objects. Bold lines show how the TPG would look like if the requested knowledge unit is granted.

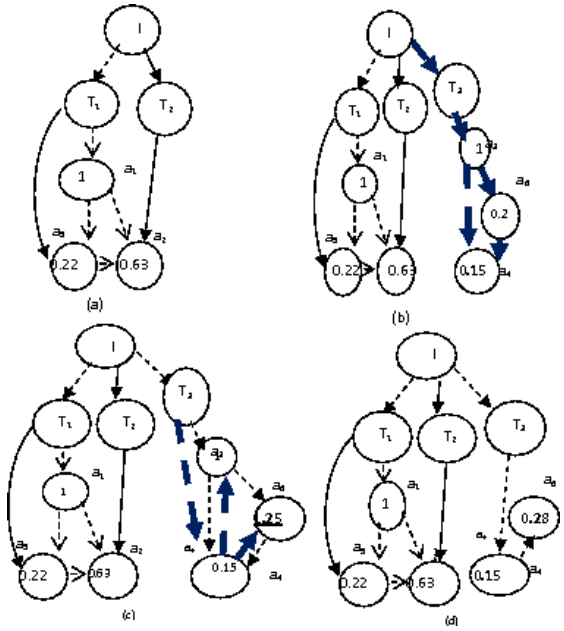

Fig. 8. Predicting and Preventing Insider Threat Using the TPG.

7 Conclusions and Future Work

This paper investigates insider threat problem in relational database systems. We consider different levels of granularities of data items and identify various types of dependencies among those data items. Insiders who have knowledge of these dependencies may infer values of data items to which they do not have authorized access. In addition, we have explained how constraints on dependencies play a role in knowledge acquisition. We have introduced the Constraint and Dependency graph (CDG) and the dependency matrix; these data structures show various types of dependencies and corresponding constraints in a relational database system. An algorithm for constructing insiders knowledge base has been provided. This algorithm helps in building the knowledge base of an insider and determines the data items which the insider can access indirectly. Moreover, we have defined another graph, called the Threat Prediction Graph (TPG), which can be used to predict and prevent insider threat in terms of getting unauthorized knowledge of data items at various levels of a relational

database. As a future work, we plan to conduct experiments to establish the effectiveness of the models.

Acknowledgements

This work has been supported in part by US AFOSR under grant FA 9550-08-1-0255. We are thankful to Dr. Robert. L. Herklotz for his support, which made this work possible.

References

1. Brackney, R., and Anderson, R. Understanding the insider threat. In: Proceedings of a march 2004 workshop. Technical report, RAND Corporation. Santa Monica, CA (2004).
2. Gordon, L., Loeb, M., Lucyshyn, W., Richardson, R. Computer Crime and Security Survey. Available at `http://www.cpppe.umd.edu/Bookstore/Documents/2005CSISurvey.pdf`.
3. Yaseen, Q., Panda, B. Knowledge Acquisition and Insider Threat Prediction in Relational Database Systems. In Proceedings of the International Workshop on Software Security Processes, pp. 450–455. Vancouver, Canada (2009).
4. Chagarlamudi, M., Panda, b., Hu, Y. Insider Threat in Database Systems: Preventing Malicious Users' Activities in Databases. In: Proceedings of the 2009 Sixth International Conference on Information Technology: New Generation, pp.1616–1620. Las Vegas (2009).
5. Bishop M., Gates, C.: Defining the Insider Threat. In Proceedings of the 4th Annual Workshop on Cyber Security and Information Intelligence Research, Vol. 288. Tennessee (2008).
6. Maybury, M., Chase, P., Cheikes, B., Brackney, D., Matznera, S., Hetherington, T., Wood, B., Sibley, C., Marin, J., Longstaff, T. Analysis and Detection of Malicious Insiders. In Proceedings of the International Conference on Intelligence Analysis. VA (2005).
7. Spitzner, L. Honeypots: Catching the Insider Threat. In Proceedings of the 19th Annual Computer Security Applications Conference. Washington (2003).
8. Althebyan, Q., Panda, B. A knowledge-base model for insider threat prediction. In Proceedings of the IEEE Workshop on Information Assurance and Security, pp. 239–246. West Point, NY (2007).
9. Farkas, C., Jajodia, S. The Inference Problem: A Survey. ACM SIGKDD Explorations, Vol. 4, pp. 6–11. (2002).
10. Farkas, C., Toland, T., Eastman, C. The Inference Problem and Updates in Relational Databases. In Proceedings of the 15th IFIP WG11.3 Working Conference on Database and Application Security, pp.181–194. (2001).
11. Brodsky, A., Farkas, C., Jajodia, S. Secure Databases: Constraints, Inference Channels and Monitoring Disclosures. In Proceedings of the IEEE Trans. on Knowledge and Data Engineering, Vol. 12, pp. 900–919. (2000).
12. Yip, R., Levitt, K. Data Level Inference Detection in Database Systems. In Proceedings of the 11th Computer Security Foundations Workshop, pp. 179-189. Rockport, MA (1998).
13. Murata, T.: Petri nets: Properties, analysis and applications. In Proceedings of the IEEE, Vol. 77, pp. 541–580. (1989).

Algorithm 1: Knowledgebase Algorithm.

Input: An insider I, Dependency Matrix, CDG, NIDG, Hot and Safe clusters, S: Set of tables that insider has direct read access.

Output: The knowledge graph of the insider *I*.

Initialize the KG=(V,E) , where V={I}, E={}, and insider *I*.
for *each table* T_k *in S //add direct access tables to the graph* **do**
    V=V∪ {T_k } //add the node T_k to KG.
    E=E∪ {e(I,T_k)} // add edge e(I, T_k) to the KG
    for *each a with attributes(*T_k*) and the insider has a read access to it // add direct access attributes to the KG* **do**
        V=V∪ {a} //add the node a to KG
        E=E∪ {e(T_k, a)} // add edge e(T_k, a) to the KG
for *each* T_k *in S* ***do*** *// consider dependencies* **do**
    for *each safe cluster R to which* T_k *belongs* **do**
        Exclude all tables in R from the knowledgebase of *I*
    for *each hot cluster H to which* T_k *belongs* **do**
        for *each table* T_m *that belongs to H//* $T_m \neq T_k$ **do**
            V=V∪ {T_m } //Add the node T_m to KG
            E=E∪ {e(I, T_m)}//add edge e(I, T_m) to the KG
        for *each a ∈ attributes(*T_m*) that the insider can deduce information about // add the attribute to the KG* **do**
            V=V∪ {a} //add the node a to KG
            E=E∪ {e(Tm, a)}//add e(Tm, a) to KG if Tm ≠ Tl
            E= E∪ {e(al, a)} // al is an attribute that belongs to a table in S and on which a depends
    for *each other table Ts that has dependency (one way) with Tk // add tables from other clusters* **do**
        Repeat steps 16 to 23 for the table Ts
    for *each table Tj that depends transitively on Tk (definition9)//transitive inference* **do**
        V=V∪ {Tj} //add the node Ts to KG
        E=E∪{e(I, Tj)}// add edge e(I, Tj) to the KG
    for *each a with attributes(Tj)that the insider can infer Information transitively about it //add the attribute to the KG* **do**
        V=V∪ {a} //add the node a to KG
        E=E∪ {e(Tj, a)}//add edge e(Tj, a) to the KG
        E= E∪ {e(al, a)}//al is an attribute that belongs to a table in KG and on which a depends directly (def. 9)
for *each edge e (Tk,a)//Tk is a table and a is a Attribute* **do**
    Weight (e(Tk,a)) = the amount of information the insider has about a // using NDIG
for *each edge e(I,Tk)* **do**
    Weight(e(I,Tk))= $\sum_{i=1}^{n}$ weight(e(Tk,ai)))/n, where n is the number of attributes in Tk.