DEMIDS: A Misuse Detection System for Database Systems

Christina Yip Chung, Michael Gertz, Karl Levitt
{chungy | gertz | levitt}@cs.ucdavis.edu
Department of Computer Science, University of California at Davis
One Shields Avenue, Davis, CA 95616-8562, USA

Abstract

Despite the necessity of protecting information stored in database systems (DBS), existing security models are insufficient to prevent misuse, especially insider abuse by legitimate users. Further, concepts for misuse detection in DBS have not been adequately addressed by existing research in misuse detection. Even though there are available means to guard the information stored in the database system against misuse, they are seldom used by security officers because security policies of the organization are either imprecise or not known at all.

This paper presents a misuse detection system called DEMIDS which is tailored to relational database systems. DEMIDS uses audit logs to derive profiles that describe typical behavior of users working with the DBS. The profiles computed can be used to detect misuse behavior, in particular insider abuse. Furthermore, the profiles can serve as a valuable tool for security re-engineering of an organization by helping the security officers to define/refine security policies and to verify existing security policies, if there are any.

Essential to the presented approach is that the access patterns of users typically form some working scopes which comprise sets of attributes that are usually referenced together with some values. DEMIDS considers domain knowledge about the data structures and semantics encoded in a given database schema through the notion of distance measure. Distance measures are used to guide the search for frequent itemsets describing the working scopes of users. In DEMIDS such frequent itemsets are computed efficiently from audit logs using the database system's data management and query processing features.

1 Motivation

In today's business world, information is the most valuable asset of organizations and thus requires appropriate management and protection. In this, database systems play a center role because they not only allow the efficient management and retrieval of huge amounts of data, but also because they provide mechanisms that can be employed to ensure the integrity of the stored data.

Reality, however, shows that such mechanisms for enforcing organizational security policies are often not adequately used. There are various reasons for this. First, security policies are often not known or not well specified, making it difficult or even impossible to translate them into appropriate security mechanisms. This observation holds for both general security policies as well as policies tailored to individual database users and applications. Second, and more importantly, security policies do not sufficiently guard data stored in a database system against "privileged users". For example, [CK96] has revealed that in computer systems the primary security threat comes from *insider abuse* rather than from *intrusion*. This observation results in the fact that much more emphasis has to be placed on internal control mechanisms of systems, such as audit log analysis.

Security models as described in, e.g., [WSF79, Den86, JS90, SW92], to *prevent misuse*[1] are insufficient to protect the information stored in database systems because of the increase in the size of data to achieve a fine grain control. More importantly, these models assume that security policies of an organization are known, which, as mentioned before, is often not the case. Misuse detection systems (MDSs)[2] are a cost-effective compromise to establish and assure a certain degree of security in a system. Nevertheless, concepts for misuse detection in database systems have not been adequately addressed by existing MDSs which neither consider the structure and semantics nor the fine granularity of data in database systems.

In this paper we propose a misuse detection system tailored to relational database systems. The system called DEMIDS (*DE*tection of *MI*suse in *D*atabase *S*ystems) provides a rich set of tools to derive user profiles from audit logs. Such profiles describe the typical behavior (access patterns) of users in the system by specifying the typical values of features that are audited in audit logs. The profiles derived are used to detect misuse behavior. Although it can be used to detect both intrusion and insider abuse, DEMIDS places emphasis on the detection of malicious behavior by legitimate users who abuse their privileges. Hence the systems is particularly useful for internal control. Our system can complement misuse detection at the operating system layer because intrusion attempts that MDSs fail to detect at the operating system layer may be detected as anomalous events at the database system layer. Further, the profiles derived can serve as a valuable tool for *security re-engineering* of an organization by helping the security officer (SSO) to define/refine security policies and to verify existing security policies, if there are any. Finally, profiles can be used to implement respective enforcing mechanisms in the database systems using, e.g., triggers, assignment of privileges, or roles.

Essential to the proposed approach is that, given a database schema and associated applications, the access patterns of users will form some *working scopes* comprising certain sets of attributes that are usually referenced together with some values. The idea of working scopes is nicely captured by the concept of *frequent itemsets* which are sets of features with certain values. Based on the data structure and semantics (integrity constraints) encoded in the data dictionary and the user behavior reflected in the audit logs, DEMIDS defines a notion of *distance measure* which measures the closeness of a set of attributes with respect to the working scopes. Distance measures are used to guide the search for frequent itemsets in the audit logs by a novel data

[1] Misuse includes both insider abuse and intrusion.

[2] Intrusion Detection System IDS is often used instead of MDS. However, the term IDS is confusing under the author's definition of intrusion and misuse. Since most systems detect both intrusion and insider abuses, we will adopt the terminology MDS.

mining approach that takes advantage of the efficient data processing functionality of database management systems. Misuse, such as tampering with the integrity of data, then can be detected by comparing the derived profiles against the security policies specified or against new information (audit data) gathered about the users.

1.1 Related Work

The security goals of database systems are availability, confidentiality and integrity [CFMS95]. Mandatory and discretionary access control models have been proposed for general computer systems to achieve these goals [BLP73, Bib77, Dio81, HRU76]. Nevertheless, these mechanisms typically operate on the file and command/process level of operating systems, which is too coarse for the finer level of granularity of data in database systems.

There are various extensions of these security models to database systems. [Den86, JS90, SW92] extend the concept of mandatory access control in relational database systems by allowing polyinstantiation of data at the tuple level. [WSF79] provides a mapping between access control in a DBS to that at operating system level. These mechanisms are fundamentally the same as general mandatory access control models and hence suffer the same limitation of being only applicable to an organization with known security policies. Further, polyinstantiations come at the cost of increasing the number of tuples in the database.

Detection mechanisms are employed to complement the shortcomings of prevention mechanisms [JV91, VL89, HDL+90, SCCC+96, FHSL96, LS98]. Nevertheless, concepts for misuse detection in database systems have not been adequately addressed by existing misuse detection systems. These systems typically reside on the operating system and/or network which work with files and system commands. The mapping between files in operating systems to relations and attributes in database systems is not exact and hence cannot closely reflect the user behavior. Moreover, auditing the user behavior at these layers is unsuited for misuse detection at the DBS level because the semantics and structure of the data are not reflected in such audit logs. Unlike previous MDSs, such domain knowledge is considered by DEMIDS to derive user profiles.

1.2 Terminology

In the rest of the paper we adopt the relational database model as the underlying data model for DEMIDS. We assume a given database schema $\mathcal{S} = \langle \mathcal{R}, \mathcal{IC} \rangle$ where $\mathcal{R}$ is a set of relation schemas and $\mathcal{IC}$ is a set of integrity constraints that have to be satisfied by every instance of the database schema. A relation schema $R \in \mathcal{R}$ is denoted by $R = \langle A_1, ..., A_n \rangle$ where each A_i is an attribute. We denote the attributes associated with a relation schema $R \in \mathcal{R}$ by $attr(R)$, and the attributes contained in the whole schema $\mathcal{S}$ by $attr(\mathcal{S})$. The value of attribute A_i of tuple t from an instance of a relation schema R is denoted by $t.A_i$

The integrity constraints considered in this paper include primary and foreign key constraints imposed on relations in $\mathcal{R}$. We furthermore assume that associated with the database is a set of applications. A database application is considered to be a sequence of (parameterized) SQL queries. Users interact with the database system through a set of operations which are either

issued by applications on behalf of the users, or directly by users in form of free form database modifications as they are typically issued by, e.g., database administrators.

1.3 Organization of the Paper

The rest of the paper is organized as follows: In Section 2, we discuss the architecture of DEMIDS, in particular its coupling with a given database system. In Section 3, we introduce the notions of distance measure and frequent itemsets to capture the working scopes of users. In Section 4, we present a novel data mining algorithm to discover profiles for users based on the idea of distance measure and frequent itemsets. In Section 5, the advantages of the Profiler are discussed and its application to some scenarios is given. We conclude the paper in Section 6 with a summary and overview of future research.

2 Architecture

The proposed misuse detection system DEMIDS is tightly coupled to an existing database system in that DEMIDS utilizes certain functionality of the system such as auditing and query processing. DEMIDS consists of four components (Figure 1): (1) Auditor (2) Data Processor (3) Profiler and (4) Detector.

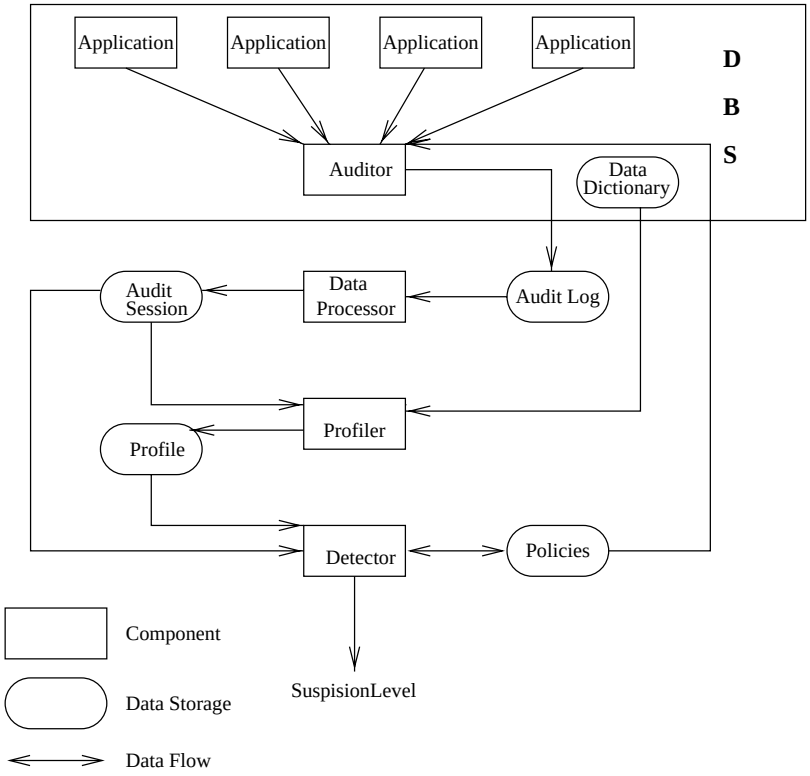

Figure 1: Components of DEMIDS's Architecture

The Auditor is responsible for collecting the audit data of users by auditing their queries through the auditing functionality of the DBMS. A set of interesting features to audit is selected by the security officer (SSO), depending on the security policy to establish or verify. For example, if a security policy states that access to a set of sensitive attributes should be monitored, an interesting feature would be the set of attributes (names) referenced in the queries. If fabrication of data is the concern, new and old values of attributes in update and insert queries should be audited. In general, we do not assume scenarios where "everything" is audited. The features that have to be audited are selected by the SSO. In practice, features are selected depending on whether the behavior and access patterns of particular users are of interest or whether the usage of certain applications by certain users is of interest.

Monitored features are recorded in audit logs. In order for the auditing and log management not to become a bottleneck of the database system and associated applications, it is possible to periodically purge audit logs to another database system which then is also used by the other components of DEMIDS.

The Data Processor is responsible for preprocessing the raw data in the audit logs, such as the handling of missing values, converting the raw data into appropriate data structures and types for the Profiler. More importantly, it groups the raw audit data into *audit sessions*. This is a critical step because the way audit data are aggregated into audit sessions determines what profiles are generated. For instance, the data can be grouped according to users or roles adopted by the users [SCFY96].[3] User profiles are generated in the former case and role profiles in the latter.

During the training stage, which is typically supervised by the SSO, the Profiler generates a profile for each audit session. The Profiler consults the repository of domain knowledge, such as the database schema, to guide its search for profiles that satisfy certain interesting measures, for example, a sufficient support. It is assumed that during the training stage, users do not perform malicious behavior to train the Profiler and that there are enough data collected to represent their typical behavior. For instance, in an office system, the training stage can include office hours during weekdays (since most business operations are performed during office hours of weekdays) and the first and last few days of a month (to cover monthly operations). In systems where users have well defined job descriptions, user behavior is fairly stable and the training stage can last for a shorter period of time, for instance, only a few days.

During the monitoring stage, the Detector computes a score to determine if user activities are suspicious. This can be achieved by comparing the new information (audit records) about user activities against the corresponding profiles derived during the training stage. Another way is to compare the user profiles against the security policy. Both the profiles and security policies in DEMIDS are specified in a rule based format. Comparison of the profiles and policies can be based on the number of new rules, missing rules and rules with the same precondition but different consequents.

[3] A role is a collection of privileges necessary to accomplish a certain task.

3 Approach

In this section, we describe the main idea behind the concept of working scopes for users. We then define the notion of distance measure to capture the degree of "closeness" of a set of attributes in a given database schema with respect to the working scopes. This is followed by a description of the concept of frequent itemsets which employs distance measures to represent the working scopes of users. The idea of frequent itemsets forms the basis for deriving user profiles.

3.1 Working Scopes

We conjecture that a user typically will not access all attributes and data in a schema and databases respectively. Attributes used in a query are related through primary and foreign key dependencies among the relations in a schema using join conditions. Therefore, the access patterns of users will form some *working scopes* which are sets of attributes that are usually referenced together with some values. A profile captures the idea of working scopes by specifying the typical values of sets of features in an audit session.

Example 3.1 We use the sample database schema shown in Figure 2 as an example.

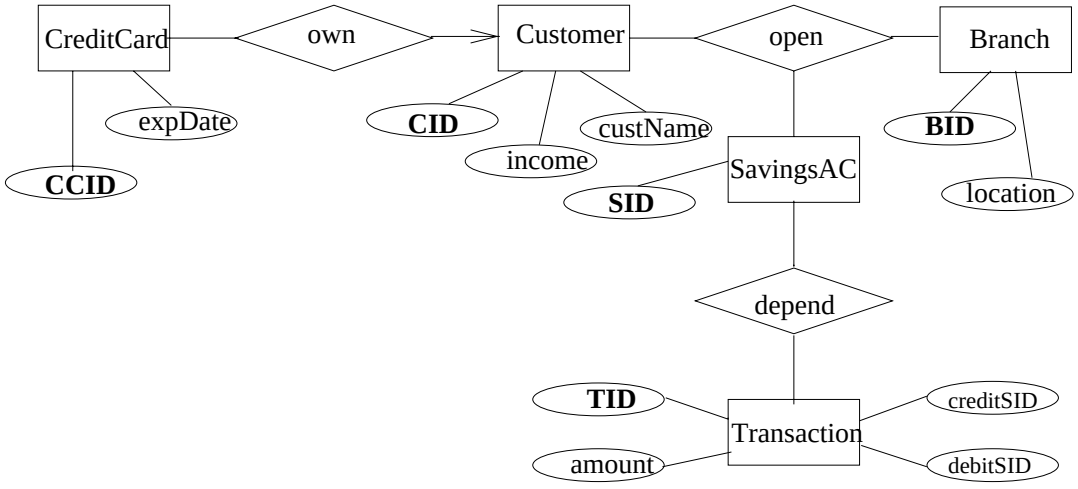

Figure 2: Sample Database Schema

This database schema can easily be derived from the information recorded in the data dictionary. Let the set of features in audit sessions be the type of the query operation (queryType), the sets of attributes referenced by the query ($R.A = 1$ if attribute A from relation R is referenced, 0 otherwise), the relation referenced by the insert query (relation), the values of attributes ($R.A.Val$), and their new values ($R.A.newVal$) for update and insert queries. Suppose user `Teller` is responsible for entering transactions by issuing database modifications of the type:

insert into transaction **values**(TID, $amount$, $debitSID$, $creditSID$)

where $TID, amount, debitSID, creditSID$ are variables.

The working scope WS of `Teller` then would be:

$WS_{\texttt{Teller}}$ = {queryType='insert', relation='transaction'}.

3.2 Distance Measure

Working scopes of users consist of attributes that are closely related in the database schema and are often referenced together in database modification statements (which, of course, can include data retrieval statements). To capture the idea of "closeness" of attributes in a database schema, we introduce the notion of *distance measure*. Our notion of distance measure is used to guide the Profiler in discovering profiles from audit sessions.

Considering a given database schema $\mathcal{S}$, attributes are structurally close if they either belong to the same relation or can be related by exploiting (a sequence of) foreign key dependencies. This aspect is reflected by the schema distance function.

Definition 3.2 (Schema Distance Function)
Assume a database schema $\mathcal{S}$ with a set $\mathcal{R}$ of relation schemas. Given two attributes $A_i \in R, A_j \in S$ where $R, S \in \mathcal{R}$, the pairwise schema distance between A_i and A_j, denoted by $PSDist(A_i, A_j)$, is defined as

$$PSDist(A_i, A_j) := \frac{ShortestDist(R, S)}{\max\{ShortestDist(R_k, R_l) \mid R_k, R_l \in \mathcal{R}\}}$$

Given a set of attributes $\{A_1, \ldots, A_n\} \subseteq attr(\mathcal{S})$, the schema distance function is defined as

$$SchemaDist(A_1, \ldots, A_n) := \max\{PSDist(A_i, A_j)\}$$

where $ShortestDist(R, S)$ computes the shortest distance between two relations R and S in the database schema based on primary and foreign keys by which R and S can be related. ■

We normalize the distance measure by the maximum shortest distance between any pair R_k, R_l of relations in the database schema so that the value of distance measure falls in the range of 0 to 1. The nearer the value of the distance measure to 1, the closer is the set of attributes.

Although two attributes are schematically close, this does not necessarily mean that they are semantically close, too. Since we would like to derive a profile for each audit session, the access patterns of the attributes in audit sessions should be considered in the distance measure as well. In order to capture this aspect, we define an access affinity function which considers the dynamic access patterns on the attributes.

Definition 3.3 (Access Affinity Function)
Given a set $\mathcal{A} = \{A_1, \ldots, A_n\} \subseteq attr(\mathcal{S})$ of attributes contained in a database schema $\mathcal{S}$. The *access affinity* of $\mathcal{A}$, denoted by $AA(\mathcal{A})$, is defined as

$$AA(\mathcal{A}) := \frac{AAC(\mathcal{A})}{\max\{AAC(A_{i_1}, \ldots, A_{i_m}) \mid \{A_{i_1}, \ldots, A_{i_m}\} \subseteq attr(\mathcal{S})\}}$$

where $AAC(A_1, \ldots, A_n)$ is the total number of audit records in the session such that each audit record references all attributes $A_1, \ldots, A_n$. ■

Based on the schema distance function and access affinity, we are now able to define a distance measure between a set of attributes that takes both structural and access properties of the attributes involved into account.

Definition 3.4 (Distance Measure)
Given a set $\mathcal{A} = \{A_1, \ldots, A_n\} \subseteq attr(\mathcal{S})$. The distance among the attributes in $\mathcal{A}$, denoted by $Dist(\mathcal{A})$, is defined as

$$SWeight * SchemaDist(A_1, \ldots, A_n) + (1 - SWeight) * (1 - AA(A_1, \ldots, A_n)) \quad ■$$

We normalize the distance measure by choosing $SWeight \in \mathbb{R}[0, 1]$. Since the domain of $SchemaDist$ and AA is $\mathbb{R}[0, 1]$, $Dist \in \mathbb{R}[0, 1]$.

$SWeight$ is a value that has to be specified by the SSO prior to the auditing and is used to weigh the schema distance component. The higher the value for $SWeight$, the more important is the schematic property in computing the distance measure, and vice versa. If users often access attributes of relations that are related by some foreign key dependencies, then $SWeight$ can be set to a higher value.

Example 3.5 We use the sample database schema shown in Figure 2 to demonstrate how our notion of distance measure reflects the working scopes of users. We consider two sets of attributes, namely $S_1 =$ {t.TID, t.amount, t.debitSID, t.creditSID} and $S_2 =$ {t.TID,c.CID}.[4] S_1 is the working scope of the user `Teller` while S_2 is an arbitrary set of attributes.

The maximum distance between any pair of relations in the schema is 6, which is the distance between attributes of relations CreditCard and Transaction. Hence we have:

$SchemaDist(S_1) = 0.0$
$SchemaDist(S_2) = 4/6 = 0.67$

Suppose in the audit session we have two audit records corresponding to the insert queries shown in Example 3.1. Hence, $AAC(t.TID, t.amount, t.debitSID, t.creditSID) = 2$ and the access affinity among the attributes in each set is

$AA(S_1) = 2/2 = 1.0$
$AA(S_2) = 0/2 = 0.0$

Suppose $SWeight = 0.5$, the distance measures of S_1, S_2 are then

$Dist(S_1) = 0.5 * 0.0 + (1 - 0.5) * (1 - 1.0) = 0.0$
$Dist(S_2) = 0.5 * 0.67 + (1 - 0.5) * (1 - 0.0) = 0.33$

The distance measure of S_1 is very small because the attributes are closely related in the database schema and are often referenced together in the queries. The distance measure for S_2 is larger because the attributes are not only further apart in the database schema, but also not referenced together in the queries.

[4] We use the notation $R.A$ to denote attribute A from relation R. We use shorthands t to denote the relation *transaction* and c for *customer*.

Since S_1 has a smaller distance measure, features referencing this set of attributes would likely fall into the same working scopes. Similarly, since S_2 has a greater distance measure, features referencing this set of random attributes would not be considered to be in the same working scope.

3.3 Frequent Itemsets

We use the concept of *frequent itemsets* to describe the working scopes of users. A frequent itemset is a set of features with values assigned to them. The set of features is selected by the SSO. For instance, the timestamp of audit records is an interesting feature if we are interested in the temporal aspect of user behavior. The sets of attributes referenced by the queries are interesting too, especially if they belong to relations that are sensitive. The new and old values of tuples of update queries are important if fabrication of data is concerned. The domain of a feature F_i is denoted by $Domain(F_i)$. For instance, if timestamp is recorded as the number of seconds elapsed since a particular moment of time, then the domain of timestamp is real numbers. The domain of userID can be strings. Since our notion of distance measure reflects the dependencies among relations as well as the access patterns of attributes in working scopes, frequent itemsets are enriched by a distance measure component to capture such knowledge.

Definition 3.6 (Frequent Itemset)
Given a set of features $\mathcal{F} = \{F_1, \ldots, F_m\}$ audited in audit session $AuditS$. An itemset I for $\mathcal{F}$ is defined as $I := \{F_1 = f_1, \ldots, F_m = f_m\}, [sup, dist]$. I is said to be a frequent itemsets in $AuditS$ if

- $F_i \in \mathcal{F}, 1 \leq i \leq m$
- $f_i \in Domain(F_i), 1 \leq i \leq m$
- $sup \equiv support(I, AuditS) \geq supThreshold$
- $dist \equiv Dist(A_1, \ldots, A_n) \leq distThreshold$

where

- $A_1, \ldots, A_n$ are corresponding attributes for features $F_1, \ldots, F_m$,
- $supThreshold$ and $distThreshold$ are user defined parameters, and
- $support(I, AuditS)$ computes the number of audit records in $AuditS$ that satisfy I.

Attributes corresponding to the features are those attributes referenced by the features. For example, if feature $R.A$ records whether attribute A from relation R is referenced in a query, then the corresponding attribute is attribute A. If feature *relation* records which relation is referenced by an insert query, then the corresponding attributes are the attributes of the relation. Some features such as userID and timestamp do not have corresponding attributes in the database schema. If the set of features do not reference any attribute, the distance measure of this frequent itemset can be defined as zero.

supThreshold can be expressed in terms of the number of audit records or in terms of percentage of audit records in the audit session. *distThreshold* is within $\mathbb{R}[0,1]$. *supThreshold* and *distThreshold* are adjusted by SSO. The higher the value of *supThreshold* and the lower the value of *distThreshold*, the more selective are the frequent itemsets. If tighter monitoring is desired, for example, during the training stage, *supThreshold* and *distThreshold* should be adjusted accordingly so that more selective frequent itemsets are discovered. Therefore, only "very" typical user behavior is described in the profiles. During the monitoring stage, *supThreshold* can be lowered while *distThreshold* is raised to discover more frequent itemsets. Mismatch between the frequent itemsets discovered in monitoring stage and those in the training stage can trigger alarm to SSO.

The frequent itemsets of a user in an audit session correspond to the profile of the user in that audit session. Audit data in the audit logs are grouped into separate audit sessions according to some properties, such as grouping under the same userID. Let $p_{session}$ be a predicate grouping the audit records in an audit session and $I = \{F_1 = f_1, \ldots, F_m = f_m\}$ a frequent itemset for the audit session. Then a corresponding profile statement in rule-based format is

$$p_{session} \rightarrow F_1 = f_1 \wedge \ldots \wedge F_m = f_m.$$

The working scopes of users are sets of attributes that are often referenced together with certain typical values. Therefore, sets of feature/value pairs can nicely represent the working scopes of users. It should be mentioned that frequent itemsets are a better representation for working scopes than clusters since objects in clusters are not tagged with values. Furthermore, it is more appropriate to use frequent itemsets to describe working scopes than to use association rules [AS94] since there is no causal relationship in the access of attributes in queries. Although the profiles for frequent itemsets are transformed to rules by the predicate describing the audit session, the working scopes of users are represented by sets of feature/value pairs.

Frequent itemsets that are subsets of other frequent itemsets represent the same working scopes. Therefore, we are interested in discovering *minimal frequent itemsets* that cover smaller frequent itemsets.

Definition 3.7 (Minimal Frequent Itemset)
Given a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of frequent itemsets in an audit session *AuditS*. A frequent itemset $I \in \mathcal{I}$ is a minimal frequent itemset in *AuditS* if there is no other frequent itemset $I' \in \mathcal{I}$ such that $I \subset I'$. ■

Example 3.8 Suppose we have a set of frequent itemsets $\mathcal{I}$=$\{I, I'\}$ for audit session *AuditS* where

I = {queryType='select', t.amount=1, t.creditSID=1 }
I' = {queryType='select', t.amount=1, t.creditSID=1, t.debitSID=1, c.custName=1 }.

I is not a minimal frequent itemset in $\mathcal{I}$ for *AuditS* because I' is a superset of I. I' is a minimal frequent itemset in $\mathcal{I}$ for *AuditS*.

Frequent itemsets are evaluated by some interesting measures, such as minimality. It is important that all frequent itemsets satisfying this measure are discovered. Therefore, we introduce the notion of completeness.

Definition 3.9 (Completeness)
A set of frequent itemsets $\mathcal{I}$ is complete in an audit session $AuditS$ if $\mathcal{I}$ contains all minimal frequent itemset in $AuditS$. ■

We described our conjecture on the access patterns of users which form some working scopes, and discussed how we can represent these working scopes by distance measures and frequent itemsets. We have also introduced the notion of minimality and completeness to evaluate the frequent itemsets discovered by the Profiler. In the next section, a data mining algorithm to discover all minimal frequent itemsets from audit logs is presented.

4 Frequent Itemsets Profiler

In this section, we present a data mining algorithm for the Frequent Itemset Profiler to discover the frequent itemsets from an audit session. The algorithm is tightly integrated with the database system by storing the data in tables and by using SQL queries to take advantage of the query processing capabilities of the DBMS.

We first describe the data structures and input to the Profiler before presenting the algorithm. We establish several criteria for an evaluation of a Profiler and show that those criteria are satisfied by the proposed Profiler.

4.1 Data Structures

We assume that data about audit sessions are stored in a table called `Session` $= \langle TID, feature, fvalue \rangle$. Each audit record is assigned a unique TID. $feature$ and $fvalue$ store the feature/value pair of an audit record.

We use the tables `F` $= \langle FIID, A, AVal \rangle$ and `FMaster` $= \langle FIID, sup, dist \rangle$ to store the itemsets. Each itemset is assigned a unique FIID. $A, AVal$ correspond to the feature/value pair of an itemset. $sup, dist$ are the support and distance measure of the itemsets. We use a separate table `FMaster` to store the support and distance measure of itemsets because it is not desirable to repeat them for each item in an itemset.

The table $L_k = \langle sup, A_1, A_1.Val, \ldots, A_k, A_k.Val \rangle$ stores the itemsets $\{A_1 = A_1.Val, \ldots, A_k = A_k.Val\}$ with support $\geq supThreshold$. Tables $L_1, \ldots, L_n$ (where n is the total number of attributes in the database schema) are used to discover all itemsets with a sufficient support. As described later, L_k is derived from L_{k-1} and hence we only need to use two tables. However, this method is not described in the algorithm for the sake of clear illustration.

Example 4.1 Table `Session` stores audit records:

queryType='select', t.TID=1, t.amount=1, t.debitSID=1, t.creditSID=0 with TID=1
queryType='select', t.TID=1, t.amount=0, t.debitSID=0, t.creditSID=1 with TID=2

Tables `F` and `FMaster` store itemsets:

{queryType='select', t.TID=1, t.amount=1, t.debitSID=1} [100, 0.1],
{queryType='select', c.income.Val='high', cc.CCID=1} [50, 0.2]

Table `Session`

TID	Feature	Fvalue
1	queryType	'select'
1	t.TID	1
1	t.amount	1
1	t.debitSID	1
1	t.creditSID	0
1	queryType	'select'
1	t.TID	1
1	t.amount	0
1	t.debitSID	0
1	t.creditSID	1

Table `F`

FIID	A	AVal
4-1	queryType	'select'
4-1	t.TID	1
4-1	t.amount	1
4-1	t.debitSID	1
3-1	queryType	'select'
3-1	c.income.Val	'high'
3-1	cc.CCID	1

Table `FMaster`

FIID	sup	dist
4-1	100	0.1
3-1	50	0.2

In the current prototype of DEMIDS, we use the Oracle8 server ([Ora97]) as our underlying DBMS to store and process the audit data.

4.2 Algorithm

The algorithm to discover all minimal frequent itemsets in an audit session is divided into four steps:

Step 1: Derive tables $L_1, \ldots, L_n$ where n is the total number of attributes in the database schema. Insert all itemsets in $L_1, \ldots, L_n$ into `F`.

Step 2: Compute distance measure for itemsets in `F`. Store them in FMaster

Step 3: Prune frequent itemsets in `F` with distance measure $> distThreshold$.

Step 4: Prune non-minimal frequent itemsets in `F`. Update `FMaster` accordingly.

Step 1

(Initialization) `F`, `FMaster` $\leftarrow \emptyset$

for (k=1;k $\leq n$; k++)
  $L_k \leftarrow \emptyset$
  (Generate L_k *from* L_{k-1}*)*
  insert into L_k
  select count(unique R_1.TID) as count,
    com.A_1, com.A_1Val,
    ...
    com.A_k, com.A_kVal
  from (**select** one.A_1 as A_1, one.A_1Val as A_1Val,
      ...
      one.A_{k-1} as A_{k-1}, one.$A_{k-1}Val$ as $A_{k-1}Val$
      two.A_{k-1} as A_k, two.$A_{k-1}Val$ as A_kVal

from L_{k-1} one, L_{k-1} two
where one.A_1=two.A_1 $\wedge$
$\ldots$
$\wedge$ one.A_{k-2}=two.A_{k-2}
$\wedge$ one.A_{k-1} < two.A_{k-1}) com,
(**select** * **from** `Session`) R_1,
$\ldots$
(**select** * **from** `Session`) R_k
where R_1.TID=R_2.TID $\wedge \ldots \wedge$ R_{k-1}.TID=R_k.TID
$\wedge$ R_1.feature=com.A_1 $\wedge$ R_1.fvalue=com.$A_1 Val$
$\ldots$
$\wedge$ R_k.feature=com.A_k $\wedge$ R_k.fvalue=com.A_kVal
$\wedge$ count $\geq$ *supThreshold*

(Insert all tuples of `L`$_k$ *into* `F`*)*
`F` $\leftarrow$ `F` $\bigcup$ **select** $A_1, A_1Val, \ldots, A_k, A_kVal$, count **from** `L`$_k$

Step 2

(Enumerate all itemsets in `F`*)*
for **each tuple** t in (**select** unique FIID **from** `F`)
(Find all items of this itemset)
S' $\leftarrow$ **select** $A, Aval$ **from** `F`
where `F`.FIID=t.FIID
(Comput support and distance measure for this itemset)
support $\leftarrow$ **select** sup **from** `L`$_k$
where $\bigwedge_{t' \in S'}$ {A_i=t'.A $\wedge A_iVal$=t'.Aval}
distance $\leftarrow$ *computeDistance*(S')
(Insert this itemset into `F` *if it is a frequent itemset)*
if (distance $\leq$ *distThreshold*)
insert into `FMaster` **values**(t.FIID,support,distance)

computeDistance(S) is a function that computes the distance measure between a set of attributes stored in table S.

Step 3

(Delete itemsets that are not frequent itemsets)
delete from `F`
where FIID not in (**select** FIID **from** `FMaster`)

Step 4

(Delete itemsets in `F` *that are subset of others)*
delete from `F`
where FIID = any
(Table one stores all itemsets that are subsets of others)
(**select** one.FIID

from `F` one, `F` two
where one.FIID < two.FIID and one.A=two.A and one.AVal=two.AVal
group by one.FIID, two.FIID
havingcount(one.FIID) =
(**select** count(*) **from** `F` **where** FIID=one.FIID))

(Update `FMaster` *accordingly)*
delete from `FMaster` **where** FIID not in (**select** FIID **from** `F`)

4.3 Analysis

Let $\mathcal{I}$ be the set of itemsets discovered by the algorithm for an audit session $AuditS$. We claim that the algorithm is *correct*, i.e. all itemsets in $\mathcal{I}$ are frequent itemsets, that the itemsets discovered are *minimal*, and that $\mathcal{I}$ is *complete* in $AuditS$.

Theorem 4.2 (Correctness)
All itemsets in $\mathcal{I}$ are frequent itemsets. That is, for each itemset $I \in \mathcal{I}$, the following two conditions hold

(a) $support(I, AuditS) \geq supThreshold$, and

(b) $Dist(I) \leq distThreshold$. ■

As shown in [AS94], itemsets of size k with high enough support can be discovered from table L_{k-1} because subsets of itemsets with support $\geq$ $supThreshold$ also have support $\geq$ $supThreshold$. Therefore, all itemsets with high enough support are inserted into F in Step 1. Since itemsets are inserted into F in Step 1 only, they satisfy condition (a). Itemsets that are not tight enough, i.e. with a too large distance measure, are pruned in Step 3. Once an itemset is deleted, it is never inserted again. Thus, they also satisfy condition (b).

Theorem 4.3 (Minimality)
All itemsets in $\mathcal{I}$ are minimal frequent itemsets for $AuditS$, that is, there are no two itemset $I, I' \in \mathcal{I}$ such that $I \neq I'$ and $I \subset I'$. ■

This property holds because all itemsets that are not minimal are pruned in Step 3 of the algorithm. Once an itemset is deleted, it is never inserted again. Hence all itemsets left are minimal.

Corollary 4.4 $\neg\exists I_i, I_j, I_k \in \mathcal{I}$ such that $I_k = I_i \cup I_j$. ■

Theorem 4.5 (Completeness)
The set $\mathcal{I}$ of itemsets is complete for $AuditS$, that is, $\mathcal{I}$ comprises all minimal frequent itemset from $AuditS$. ■

The set of itemsets discovered is complete in the audit session because, as aforementioned, all candidate itemsets with high enough support are inserted into `F` in Step (1), and an itemset is deleted from `F` only if it is not a frequent itemset (Step 3) or if it is not minimal (Step 4).

5 Discussion

5.1 Comparison

Our Frequent Itemsets Profiler is a novel approach for deriving user profiles. [AS94] proposed the concept of association rules to represent profiles of users. Algorithms based on association rules attempt to discover the causal relationships among items in transactions (which, in our case would be audit sessions). However, association rules are inappropriate to represent the working scopes of users in database systems because there is no such causal relationship in the access patterns of attributes in a query. Post-processing is necessary to prune those association rules that represent the same working scope. Unlike association rules algorithms, our Frequent Itemsets Profiler avoids discovering redundant rules that represent the same working scopes since frequent itemsets are sets of feature/value pairs.

Another related approach, the clustering approach [Eve73], discovers sets of objects, so-called *clusters*, that are close under a certain distance measure. However, the set of objects discovered are not tagged with values and hence clusters cannot represent typical values of features in working scopes. In addition, most clustering algorithms are not scalable to the size of the data. Unlike clustering algorithms, our Frequent Itemsets Profiler discovers frequent itemsets by using the data processing capability of the DBMS and hence is scalable to the size of audit data. The algorithm to discover frequent itemsets exhibits similar features as hierarchical clustering algorithm, which is illustrated by Corollary 4.4. Hierarchical clustering algorithms continuously merge smaller sets of objects into larger clusters step by step. Our algorithm takes advantage of the data processing power of DBMS by pruning non-minimal frequent itemsets in one delete query.

More importantly, we use the notion of distance measure to capture the domain knowledge encoded in a database schema, and to guide the search of interesting frequent itemsets. Since consecutive queries may correspond to similar tasks, they can be aggregated together. For example, consecutive audit records of the same query type within a certain time window can be aggregated into one audit record. The notion of distance measure then would be useful in identifying sets of attributes that correspond to the working scopes of users for these queries.

5.2 Scenarios

Here we give two scenarios to illustrate the effectiveness of our Frequent Itemsets Profiler. We use the example described in Section 3.1. Suppose user `Teller` issues the insert query often enough during the training stage. A corresponding frequent itemset discovered for `Teller` would be:

$$I_{Normal} = \{queryType =' insert', relation =' transaction'\}$$

Scenario 1: In the first scenario, suppose in an audit session, `Teller` misuses his privileges to steal credit card information about the customers by issuing the query

select ccID, expDate, C.custName
from CreditCard CC, own O, Customer C
where C.CID = O.CID and O.CID = CC.CID

A corresponding frequent itemset discovered by DEMIDS the would be

$I_{Misuse1}$ = {queryType='select', cc.CID=1, o.CID=1, c.CID=1, cc.CCID=1, cc.expDate=1}

This change of interest of `Teller` at the schema level is illustrated by the difference between the set of attributes occuring in the frequent itemset $I_{Misuse1}$ and that in I_{Normal}.

It is worth mentioning that *supThreshold* can be set to a higher level during the training stage so that only frequent itemsets corresponding to typical user behavior are discovered. In case user `Teller` only issues the misuse query infrequently and the detection of such abuse is required, the threshold can be lowered during the monitoring stage to detect those infrequent queries. Mismatch of this outliner behavior against the typical behavior detected by the Detector can trigger alarm to the SSO.

Scenario 2: The second scenario involves a finer level of granularity of misuse. Suppose `Teller` does not change the set of attributes he usually references. He tries to transfer money from other accounts to his account *badAccount* illegally by issuing the following query very often in an audit session:

insert into transaction **values** (TID, $amount$, $debitSID$, $badAccount$)[5]

A frequent itemset discovered by the Profiler can be:

$I_{Misuse2} = \{queryType =' insert', relation =' transaction', transaction.creditSID.newVal =' badAccount'\}$

Frequent itemset $I_{Misuse2}$ consists of the same set of attributes as frequent itemset I_{Normal}. But there is an additional piece of information - the credit account is often *badAccount*. This represents a change of interest of `Teller` at the tuple level, which again can trigger alarm.

6 Conclusions and Future Work

In this paper we have presented the concepts and architecture underlying DEMIDS, a misuse detection system for relational database systems. DEMIDS provides security officers a means to derive user profiles from audit logs recording various features of accesses to the databases system through users and applications. The derived user profiles describe the typical user behavior in terms of typical access patterns against certain information structures (relations, attributes, and data) of a database. Derived profiles provide a security officer with a means not only to verify/refine existing security policies, but also to establish security policies as part of the security re-engineering of a given database system.

In particular, DEMIDS considers the data structure and semantics specified in the database schema through the notion of distance measure. Such domain knowledge is used to guide the Frequent Itemsets Profiler to search for frequent itemsets which can effectively represent the working scopes of users. Our Frequent Itemsets Profiler is capable of discovering all those minimal frequent itemsets in audit sessions by taking advantage of the query processing power of the DBMS. We have illustrated the effectiveness of our Frequent Itemsets Profiler in detecting misuse by several scenarios.

[5] $TID, amount, debitSID$ are variables

We have conducted an evaluation of our Frequent Itemsets Profiler based on synthesized data. We are in the process of acquiring medical and financial data (as well as underlying database schemas and associated applications) to conduct further analysis and would like to conduct analytical analysis on the performance of the Profiler. Effectiveness of the Detector can be evaluated by asking a user knowing the security policies to attempt to defeat the system by acquiring a treasure buried in the database.

One of our future research directions is to investigate other means to define the notion of distance measure, such as defining distance measures among attribute values, and using a different formulation other than the linear relationships encoded in foreign key dependencies.

We are also interested in considering other domain knowledge to guide the discovery of profiles. Groups of values for features can be replaced by some other values of higher level of abstraction. For instance, scores of range 10-20, 20-50, 50-90, 90-100 can be replaced by 'very low', 'low', 'high', 'very high' respectively. Introducing a certain degree of imprecision to the feature values helps to reveal regularities in the data. Such classification can be obtained from the SSO, or can be derived by considering the statistical distribution of feature values in an audit session.

Another interesting research issue is to derive profiles for roles. A user may perform different and unrelated tasks during his/her interaction with the database system, but roles are more closely tied to the functions or tasks performed. Role profiles may give rise to more regular patterns than user profiles since functions or tasks operate on data that are related and there is a more static set of sequences of operations. The challenge is to identify portions of the audit data that correspond to the same role. If the roles are not known to the SSO but users execute scripts on a regular basis to perform routine tasks, the scripts would serve to identify the roles the users take. In case that the user interacts with the database system through some application such as forms and reports, these applications perform well-defined database modifications on behalf of the user, and thus can be the basic units to identify the roles.

References

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, (editors), *Proceedings of the 20th VLDB Conference*, 487–499, 1994. Morgan Kaufmann Publishers.

[Bib77] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, MITRE Corp., Redford, MA, 1977.

[BLP73] D. E. Bell and L.J. La Padula. Secure computer systems: mathematical foundations. Technical Report ESD-TR-73-278, MITRE Corp., Redford, MA, Nov 1973.

[CFMS95] Silvana Castano, Maria Grazia Fugini, Giancario Martella, and Pierangela Samarati. *Database Security*. Addison-Wesley, 1995.

[CK96] Carter and Katz. Computer crime: an emerging challenge for law enforcement. *FBI Law Enforcement Bulletin*, 1–8, December 1996.

[Den86] Dorothy E. Denning et. al. Secure distributed data view: security policy and interpretation for class A1 multilevel secure relational database system. Technical Report A002, SRI International, 1986.

[Dio81] L.C. Dion. A complete protection model. In *Proceedinges of the IEEE Symposium on Research in Security and Privacy* [OAK], 49–55, 1981.

[Eve73] Brian Everitt. *Cluster Analysis*. John Wiley & Sons - New York, 1973.

[FHSL96] Stephanie Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for unix processes. In *Proceedinges of the IEEE Symposium on Research in Security and Privacy* [OAK], 120–128, 1996.

[HDL+90] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. [OAK], 296–304, 1990.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of ACM*, 19(8):461–471, August 1976.

[JS90] S. Jajodia and R. Sandhu. Polyinstantiation integrity in multilevel relations. In *Proceedinges of the IEEE Symposium on Research in Security and Privacy* [OAK], pages 104–115, 1990.

[JV91] H. Javitz and A. Valdez. The SRI IDES statistical anomaly detector. [OAK], pages 316–326, 1991.

[LS98] Wenke Lee and Salvatore J. Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium (SECURITY-98)*, pages 79–94, Berkeley, January 26–29 1998. Usenix Association.

[OAK] Proceedings of the IEEE symposium on research in security and privacy.

[Ora97] Oracle8 Server Concepts, Release 8.0. Part No. A54643-01, Oracle Corporation, Redwood City, California, 1997.

[SCCC+96] Stuart Staniford-Chen, Steven Cheung, Richard Crawford, Mark Dilger, Jeremy Frank, James Hoagland, Karl Levitt, Christopher Wee, Raymond Yip, and Dan Zerkle. GrIDS-A graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[SCFY96] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.

[SW92] K. Smith and M. Winslett. Entity modelling in the MLS relational model. In *Proceedings of the International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada, 1992.

[VL89] H. S. Vaccaro and G. E. Liepins. Detection of anomalous computer session activity. In *Proceedinges of the IEEE Symposium on Research in Security and Privacy* [OAK], pages 280–289, 1989.

[WSF79] C. Wood, R. C. Summers, and E.B. Fernandez. Authorization in multilevel database models. *Information Systems*, 4(2):155–161, 1979.