DEMIDS: A Misuse Detection System for Database Systems

Christina Yip Chung, Michael Gertz, Karl Levitt
Department of Computer Science, University of California at Davis
One Shields Avenue, Davis, CA 95616-8562, USA
Phone/Fax: +1-530-762-6468/-4767
e-mail: {chungy|gertz|levitt}@cs.ucdavis.edu

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 a 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 in queries. 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 data management and query processing features of the database management system.

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.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35501-6_14
M. E. van Biene-Hershey et al. (eds.), *Integrity and Internal Control in Information Systems*

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". (Carter and Katz 1996) 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., (Wood *et al.* 1979, Denning *et al.* 1986, Jajodia and Sandhu 1990, Smith and Winslett 1992), to *prevent misuse* (Misuse includes both insider abuse and intrusion.) 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)* 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 system 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 security officers (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

*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.

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 in a query. The idea of working scopes is conceptually 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 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 users.

1.1 Related Work

The security goals of database systems are availability, confidentiality and integrity (Castano *et al.* 1995). Mandatory and discretionary access control models have been proposed for general computer systems to achieve these goals (Bell and LaPadula 1973, Biba 1977, Dion 1981, Harrison *et al.* 1976). 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. (Denning *et al.* 1986, Jajodia and Sandhu 1990, Smith and Winslett 1992) extend the concept of mandatory access control in relational database systems by allowing polyinstantiation of data at the tuple level. (Wood *et al.* 1979) provides a mapping between access control in a DBS to that at operating system level. These mechanisms are essentially based on the 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 (Javitz and Valdez 1991, Vaccaro and Liepins 1989, Heberlein *et al.* 1990, Staniford-Chen *et al.* 1996, Forrest *et al.* 1996, Lee and Stolfo 1998). Nevertheless, concepts for misuse detection in database systems have not been adequately addressed by existing approaches to 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 semantic integrity constraints that have to be satisfied by every instance of the database schema. A relation schema $R \in \mathcal{R}$ with attributes $A_1, \ldots, A_n$ is denoted by $R = \langle A_1, \ldots, A_n \rangle$. We denote the attributes associated with a relation schema R by $attr(R)$, and the attributes of all relations in the 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 SQL queries, in particular 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 Profiler using a novel data mining algorithm to discover profiles for users based on the ideas 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.

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 SSO, depending on the security policies to establish or verify. For example, if a security policy states that access to a set of sensitive attributes should be monitored, a set of interesting features 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, insert and delete 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.

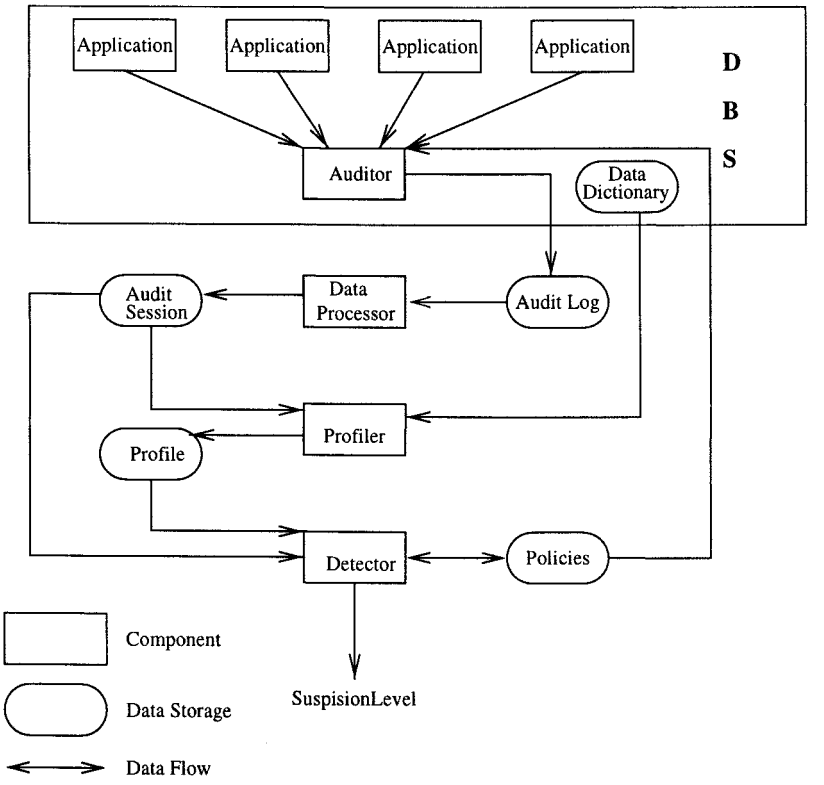

Figure 1 Components of DEMIDS's Architecture

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 used by the other components of DEMIDS.

The Data Processor is responsible for preprocessing the raw data in the audit logs, such as handling 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 (a role being a collection of privileges) adopted by the users (Sandhu *et al.* 1996). 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 policies. 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 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 measure 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 typically

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 1 We use a sample database schema shown in Figure 2 as a working example.

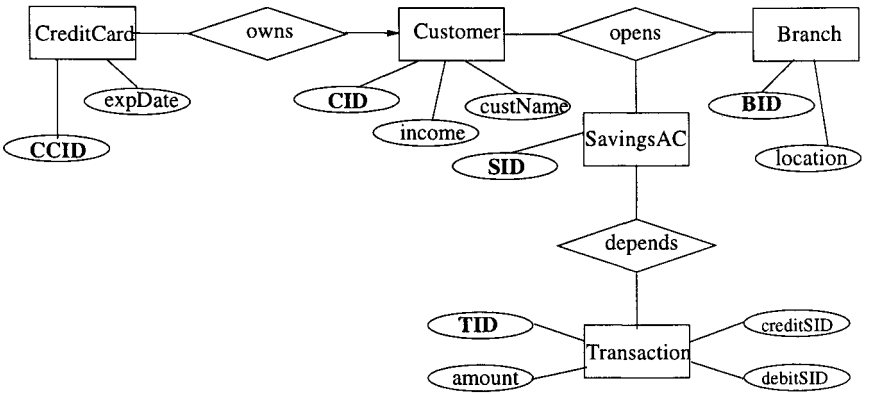

Figure 2 Sample Database Schema

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, the set of attributes referenced by the query, the relation referenced by the insert and delete query, the old and new values of the attributes. In this paper we will adopt the convention of using queryType to denote the query operation, R.A=1 the fact that attribute R.A is referenced in the query, R.A=0 the fact that R.A is not referenced, relation the name of the relation referenced, R.AVal the (old) value of R.A, and R.AnewVal the new value of R.A if the query is a data modification query. 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 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 retrieval and modification statements. To capture the idea of "closeness" of attributes in a database schema, we introduce the notion of *distance measure* which 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.

$ShortestDist(R, S)$ 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.

Definition 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 $\mathcal{A} = \{A_1, \ldots, A_n\} \subseteq attr(\mathcal{S})$, the schema distance function is defined as

$$SchemaDist(\mathcal{A}) := \max\{PSDist(A_i, A_j) \mid A_i, A_j \in \mathcal{A}\}$$

■

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 0, the closer is the set of attributes. Attributes of the same relation have the schema distance 0.

Two attributes being schematically close does not necessarily imply that they are semantically close. 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 (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 $\mathcal{A}$ in an audit session, denoted by $Aff(\mathcal{A})$, is defined as

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

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

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

$$Dist(\mathcal{A}) := SWeight * SchemaDist(\mathcal{A}) + (1 - SWeight) * (1 - Aff(\mathcal{A}))$$

■

We normalize the distance measure by choosing $SWeight \in \mathbb{R}[0, 1]$. Since the domain of $SchemaDist$ and Aff 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 5 We use the sample database schema shown in Figure 2 to demonstrate how the notion of distance measure reflects the working scopes of users. Let us consider a `Teller` who often issues the following query as part of his daily routine:

select SavingsAC.SID
from SavingsAC s, opens o, Customer c
where s.SID=o.SID ∧ o.CID=c.CID ∧ c.custName='John Smith'

The corresponding working scope is S_1 = {s.SID=1, o.SID=1, c.SID=1, c.custName=1, c.custNameVal= 'John Smith'}.

Now consider a random set of attributes S_2 = {t.TID, c.CID}. This set of attributes would not be discovered as the working scope because of its large distance measure as explained below. 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:

$SchemaDist(S_1) = 2/6 = 0.33$
$SchemaDist(S_2) = 4/6 = 0.67$

Suppose in the audit session, `Teller` issues the query in the above example 100 times. Then, $Aff(S_1) = 100/100 = 1$ and $Aff(S_2) = 0/100 = 0.0$.

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

$$Dist(S_1) = 0.5 * 0.33 + (1 - 0.5) * (1 - 1.0) = 0.17$$
$$Dist(S_2) = 0.5 * 0.67 + (1 - 0.5) * (1 - 0.0) = 0.83$$

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 are also never referenced together in the above query.

Since S_1 has a smaller distance measure, features corresponding to this set of attributes would likely fall into the same working scope. On the contrary, since S_2 has a greater distance measure, features corresponding to this set of attributes would not be considered to belong to this working scope.

The working scopes discovered can serve as a valuable tool to establish security policies. For example, a policy can state that the working scopes of users in the monitoring stage cannot deviate very much from their working scopes or the working scopes of users with similar behavior during the training stage. If a working scope of a user reveals that all the attributes of a relation are typically referenced together, a policy can be established to state that the user typically accesses that relation instead of enumerating all of its attributes.

3.3 Frequent Itemsets

We use the concept of *frequent itemsets* to describe the working scopes of users. Since the notion of distance measure reflects the dependencies among relations as well as the access patterns of attributes in working scopes, our notion of frequent itemsets is enriched by a distance measure component to capture such knowledge. 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 timestamps 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 data modification 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 a timestamp is recorded as the number of seconds elapsed since a particular moment of time, then the domain of timestamps is real numbers. The domain of userID can be strings.

Definition 6 (Frequent Itemset)
Given a set of features $\mathcal{F} = \{F_1, \ldots, F_m\}$ audited in an 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 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 of the feature is $R.A$. If feature *relation* records which relation is referenced by an insert query, then the corresponding attributes of the feature 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 the 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 indicates suspicious activities and can be brought to the attention of the 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 (Agrawal and Srikant 1994) 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 *maximal frequent itemsets* that cover the maximum number of feature/value pairs.

Definition 7 (Maximal 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 maximal frequent itemset in *AuditS* if there is no other frequent itemset $I' \in \mathcal{I}$ such that $I \subset I'$. ■

Example 8 Suppose we have a set of frequent itemsets $\mathcal{I}=\{I, I'\}$ in an 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 maximal frequent itemset in $\mathcal{I}$ for *AuditS* because I' is a superset of I. I' is a maximal frequent itemset in $\mathcal{I}$ in *AuditS*.

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

Definition 9 (Completeness)
A set of frequent itemsets $\mathcal{I}$ is complete for an audit session *AuditS* if $\mathcal{I}$ contains all maximal 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 maximality and completeness to evaluate the frequent itemsets discovered by the Profiler. In the next section, a data mining algorithm to discover all maximal frequent itemsets from audit logs is presented.

4 FREQUENT ITEMSETS PROFILER

In this section we present a data mining algorithm for the Profiler to discover frequent itemsets from an audit session. Our 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 tuple identifier 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 tuple identifier 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 $L_k = \langle FIID, sup, dist, A_1, A_1Val, \ldots, A_k, A_kVal \rangle$ stores the itemsets $\{A_1 = A_1.Val, \ldots, A_k = A_k.Val\}$ with $sup \geq supThreshold$ and distance $dist$. 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. L_k is derived from L_{k-1} and hence we only need to use two tables, namely `LNEW` and `LOLD`. The table `statItems` = $\langle A \rangle$ is a temporary table which stores a set of attributes.

Example 10 Assume table `Session` stores the following two audit records:

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

Tables `F` and `FMaster` store the itemsets (c ≡ Customer, cc ≡ CreditCard):

{queryType='select', t.TID=1, t.amount=1, t.debitSID=1} [100, 0.1],
{queryType='select', c.incomeVal= 1000, 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.incomeVal	1000
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 (Oracle 1997) as our underlying DBMS to store and process the audit data.

4.2 Algorithm

The algorithm to discover maximal frequent itemsets of size k in an audit session is divided into five steps:

Step 1: Derive table LNEW from LOLD.
Step 2: Compute distance measure for itemsets in LNEW.
Step 3: Prune frequent itemsets in LNEW with distance measure > *distThreshold*.
Step 4: Insert frequent itemsets into F.
Step 5: Delete non-maximal frequent itemsets from F. Update FMaster accordingly.

Step 1 *(Initialization)*

swap LNEW with LOLD
LNEW $\leftarrow \emptyset$
(*Generate* LNEW *from* LOLD)
insert into LNEW
select 'NA', count(unique R_1.TID) as sup, -1,
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 LOLD one, LOLD two
where one.A_1=two.A_1 $\wedge$...
$\wedge$ 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_1Val $\wedge$ ...
$\wedge$ R_k.feature=com.A_k $\wedge$ R_k.fvalue=com.A_kVal
$\wedge$ sup >= *supThreshold*

Step 2 *(Enumerate all itemsets in LNEW)*

```
fiid = 0;
for each tuple t in (select * from LNEW)
    delete from statItems
    for (int i=1;i ≤ k;i++)
        insert into statItems values(t.A_i)
        distance = select max(getDistance(one.A,two.A))
                   from statItems one, statItems two
                   where one.A ≤ two.A
        fiid = fiid + 1
        update LNEW set FIID=fiid, dist=distance
                   where (A_1 = t.A_1 ∧ ... ∧ A_nVal = t.A_nVal)
```

$getDistance : attr(\mathcal{S}) \times attr(\mathcal{S}) \rightarrow \mathbb{R}$ is a function that computes the distance measure between two attributes.

Step 3

(Delete itemsets that are not frequent itemsets)
delete from LNEW **where** dist > $distThreshold$

Step 4

```
for (int i=1;i ≤ k;i++)
    insert into F select FIID, A_i, A_iVal from LNEW
```

Step 5

```
delete from F
where FIID=any
        (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
        having count(one.FIID)= (select count(*) from F
                                 where FIID=one.FIID))

insert into FMaster
select FIID, sup, dist from LNEW where FIID 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 *maximal*, and that $\mathcal{I}$ is *complete* in $AuditS$.

Theorem 11 (Correctness)
All itemsets in $\mathcal{I}$ are frequent itemsets. That is, for each itemset $I \in \mathcal{I}$, the following two conditions hold

1. $support(I, AuditS) \geq supThreshold$, and
2. $Dist(I) \leq distThreshold$. ■

As shown in (Agrawal and Srikant 1994), 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 12 (Maximality)
All itemsets in $\mathcal{I}$ are maximal frequent itemsets in $AuditS$, 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 maximal are pruned in Step 5 of the algorithm. Once an itemset is deleted, it is never inserted again. Hence all itemsets left are maximal.

Corollary 13 $\neg\exists I_i, I_j, I_k \in \mathcal{I}$ such that $I_k = I_i \cup I_j$. ■

Theorem 14 (Completeness)
The set $\mathcal{I}$ of itemsets is complete in $AuditS$, that is, $\mathcal{I}$ comprises all maximal 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 maximal (Step 4).

5 DISCUSSION

5.1 Comparison

Our Profiler is a novel approach for deriving user profiles. (Agrawal and Srikant 1994) 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 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, the Profiler avoids discovering redundant rules that represent the same working scope since frequent itemsets are *sets* of feature/value pairs.

Another related approach, the clustering approach (Everitt 1973), 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 data. Unlike clustering algorithms, the 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 13. Hierarchical clustering algorithms continuously merge smaller sets of objects (i.e. non-maximal frequent itemsets) into larger clusters step by step. Our algorithm takes advantage of the data processing power of DBMS by pruning non-maximal frequent itemsets in one delete query in Step 5.

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. For example, consecutive queries may correspond to similar tasks and therefore can be aggregated. 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 example scenarios to illustrate the effectiveness of the 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 customers by issuing the query

select cc.CCID, cc.expDate, c.custName
from CreditCard cc, owns o, Customer c
where c.CID = o.CID and o.CCID = cc.CCID

A corresponding frequent itemset discovered by DEMIDS would be

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

This change of interest of `Teller` at the schema level is illustrated by the difference between the set of attributes occurring in the frequent itemset $I_{Misuse1}$ and that in I_{Normal}. Differences between the frequent itemsets of the training and monitoring stages can be measured based on difference in the number of missing features, the number of new features and the number of features with different values in the frequent itemsets of the training and monitoring stages.

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 an abuse is required, the threshold can be lowered during the monitoring stage to detect those infrequent queries. Mismatch of this outlier behavior against the typical behavior detected by the Detector can be brought to the attention of 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'*) where TID, *amount*, *debitSID* are variables.
A frequent itemset discovered by the Profiler can be:

$I_{Misuse2}$ = {queryType='insert', relation ='transaction',
transaction.creditSIDnewVal='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 security officers 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 Profiler to search for frequent itemsets which can effectively represent the working scopes of users. The Profiler is capable

of discovering all those maximal frequent itemsets in audit sessions by taking advantage of the query processing power of the DBMS. We have illustrated the effectiveness of the Profiler in detecting misuse by some scenarios.

We have conducted an evaluation of the 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. We would also 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 user roles.

REFERENCES

R. Agrawal, R. Srikant (1994): Fast algorithms for mining association rules. In J. Bocca, M. Jarke, C. Zaniolo (eds.), *Proceedings of the 20th VLDB Conference.* Morgan Kaufman Publishers, 487–499.

K. J. Biba (1977): Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, MITRE Corp., Redford, MA.

D. E. Bell, L.J. LaPadula (Nov, 1973): Secure computer systems: mathematical foundations. Technical Report ESD-TR-73-278, MITRE Corp.,

Redford, MA.

S. Castano, M.G. Fugini, G. Martella, P. Samarati (1995): *Database Security.* Addison-Wesley.

Carter, Katz (Dec, 1996): Computer crime: an emerging challenge for law enforcement. FBI Law Enforcement Bulletin, 1–8.

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

L.C. Dion (1981): A complete protection model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 49–55.

B. Everitt (1973): *Cluster Analysis.* John Wiley & Sons - New York.

S. Forrest, S. A. Hofmeyr, A. Somayaji, T. A. Longstaff (1996): A sense of self for unix processes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 120–128.

L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, D. Wolber (1990): A network security monitor. In *Proceedings of the IEEE symposium on research in security and privacy*, 296–304.

M. A. Harrison, W. L. Ruzzo, J. D. Ullman (Aug, 1976): Protection in operating systems. *Communications of ACM*, 19(8):461–471.

S. Jajodia, R. Sandhu (1990): Polyinstantiation integrity in multilevel relations. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 104–115.

H. Javitz, A. Valdez (1991): The SRI IDES statistical anomaly detector. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 316–326.

W. Lee, S. J. Stolfo (1998): Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium (SECURITY-98)*, 79–94, Berkeley. Usenix Association.

Oracle8 Server Concepts, Release 8.0. (1997) Part No. A54643-01, Oracle Corporation, Redwood City, California.

S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, D. Zerkle (1996): GrIDS-A graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference.*

R. Sandhu, E. Coyne, H. Feinstein, C. Youman (1996): Role-based access control models. *IEEE Computer*, 29(2):38–47.

K. Smith, M. Winslett (1992): Entity Modeling in the MLS relational model. In *Proceedings of the International Conference on Very Large Data Bases*, Vancouver, British Columbia, Canada.

H. S. Vaccaro, G. E. Liepins (1989): Detection of anomalous computer session activity. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 280–289.

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