CERIAS Tech Report 2005-70

INTRUSION DETECTION IN RBAC-ADMINISTERED DATABASES

by Elisa Bertino, Ashish Kamra, Evimaria Terzi, Athena Vakali

Center for Education and Research in
Information Assurance and Security,
Purdue University, West Lafayette, IN 47907-2086

Intrusion Detection in RBAC-administered Databases *

Elisa Bertino
Purdue University
bertino@cs.purdue.edu

Ashish Kamra
Purdue University
akamra@ecn.purdue.edu

Evimaria Terzi
University of Helsinki
terzi@cs.helsinki.fi

Athena Vakali
Aristotle University
avakali@csd.auth.gr

Abstract

A considerable effort has been recently devoted to the development of Database Management Systems (DBMS) which guarantee high assurance security and privacy. An important component of any strong security solution is represented by intrusion detection (ID) systems, able to detect anomalous behavior by applications and users. To date, however, there have been very few ID mechanisms specifically tailored to database systems. In this paper, we propose such a mechanism. The approach we propose to ID is based on mining database traces stored in log files. The result of the mining process is used to form user profiles that can model normal behavior and identify intruders. An additional feature of our approach is that we couple our mechanism with Role Based Access Control (RBAC). Under a RBAC system permissions are associated with roles, usually grouping several users, rather than with single users. Our ID system is able to determine role intruders, that is, individuals that while holding a specific role, have a behavior different from the normal behavior of the role. An important advantage of providing an ID mechanism specifically tailored to databases is that it can also be used to protect against insider threats. Furthermore, the use of roles makes our approach usable even for databases with large user population. Our preliminary experimental evaluation on both real and synthetic database traces show that our methods work well in practical situations.

1. Introduction

Data represent today an important asset. We see an increasing number of organizations that collect data, very often concerning individuals, and use them for various purposes, ranging from scientific research, to demographic trend analysis and marketing purposes. Organizations may also give access to their data, or even release such data to third parties. The increasing number of data sets being available, poses serious threats against the privacy of individuals and organizations. Since privacy is today an important concern, several research efforts have been devoted to address issues related to the development of privacy-preserving data-management techniques, such as anonymization [21] and other privacy-preserving data-mining techniques [11, 23]. Those techniques however mainly aim at modifying raw data before releasing them to other parties. The problem of developing Database Management Systems (DBMS) with high-assurance privacy and confidentiality guarantees is not that trivial ([4]). To start with, it requires a revision of architectures and techniques adopted by current DBMS ([3]). In [3] it is also pointed out that an important component of the new generation security-aware DBMS is an Intrusion Detection (ID) mechanism. ID systems specific to DBMS have not been much investigated before, whereas several ID approaches for networks and operating systems exist. We believe that there are two important reasons that motivate the development of DBMS-specific ID systems. The first is that actions malicious for a database application are not necessarily malicious for the network or the operating system; thus ID systems specifically designed for the latter would not be effective for database protection. The second, and more relevant motivation, is that ID systems designed for networks and operating systems are not adequate to protect databases against insider threats, which is an important issue when dealing with privacy. These threats are much more difficult to defend against, because they are from subjects that are legitimate users of the system, and thus may have access rights to data and resources. As an example, suppose that a clerk in a hospital usually accesses the tables and corresponding attributes containing the addresses of patients to whom billing

*This material is based upon work supported by the National Science Foundation under Grant No. 0430274 and the sponsors of CERIAS.

information need to be send. Suppose now, that suddenly this clerk issues a query accessing all attributes from the relevant tables and retrieving all patient addresses at once. Such a behavior, being so different from the normal behavior, should at least raise an alarm. We believe that building accurate profiles of legitimate user behavior and checking user access patterns against such profiles guarantees better control on the data usage, and it is a fundamental building block of any privacy solution. In this paper we address this problem by proposing a DBMS-specific ID system. It is important to note that our approach is based on the assumption that these DBMS require a high degree of assurance and security and, hence, they have well defined usage and access control policies in place. In the absence of such policies, like in the case when all users are allowed to randomly access every part of the database, the anomalies will be not well-defined and therefore, non-trivial to detect.

1.1. Our Approach

One important fact that needs to be considered is that databases typically have very large number of users. Thus, keeping a profile for each single user is not feasible in practice. We have thus based our approach on the well-known role-based access control (RBAC) model; we build a profile for each role and check the behavior of each role with respect to such profile. The RBAC model is widely used for access-control management, both in closed and open systems [10]. Authorizations are specified with respect to roles and not with respect to individual users. One or more roles are assigned to each user and privileges are assigned to roles. Managing a few roles is much more efficient than managing many individual users. With respect to ID, using roles means that the number of profiles to build and maintain is much smaller than those one would need when considering individual users. Other important advantages of RBAC are that it has been standardized (see the NIST model [19]) and has been adopted in various commercial DBMS products as well in security enterprise solutions [12]. This implies that an ID solution, based on RBAC, could be deployed very easily in practice.

In the paper, we develop our ID solution for RBAC database systems and in this context, the problem we address is as follows: how to build and maintain role profiles representing accurate and consistent user behavior; and how to use these profiles for the intrusion detection task at hand. The approach we follow relies on the use of intrusion-free database traces to extract user behavior. We build role profiles using a classifier. This classifier is then used for detecting anomalous behavior.

The main challenge in attacking the problem is to extract the right information from the database traces so that accurate profiles can be built. To address this problem, we propose several representations for the database log records. Each such representation is characterized by different granularity and correspondingly, by different accuracy levels. By using those representations, we then address our problem as a standard classification problem.

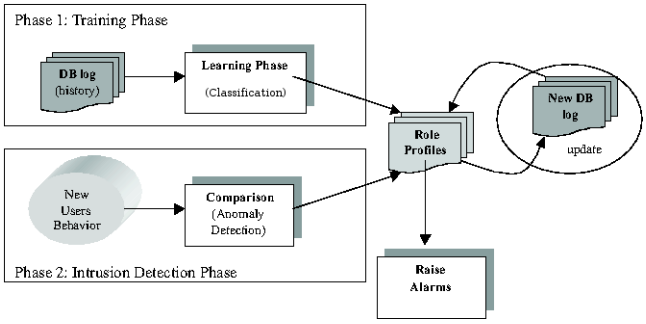

Figure 1. Overview of the ID process

1.2. System Architecture

The system's architecture consists of four main components: the user that enters queries, the conventional DBMS mechanism that handles the query evaluation process, the database log files and the ID mechanism. The latter three components constitute the new extended DBMS that is enhanced with an independent ID system, operating at the database (application) level. The flow of the interactions are as follows (Figure 1). Every time a query is issued, the database log files are updated. In the training phase, the intrusion detection system mines the existing log files and forms role profiles. In the detection phase, for every new query, the ID mechanism checks the query statement to determine whether it is anomalous. If this is the case, an alarm is raised. It should be noted that the role profiles are periodically updated (forming the New DB log in Figure 1), so that they represent the most current and relevant role behavior and false alarms are minimized.

1.3. Related Work

As we already mentioned, several approaches to ID at the operating system and network level have been developed [5, 1, 13, 17, 22]. In addition to that, some schemes have been proposed in the past related to developing an Intrusion Detection mechanism for databases. Zhu et al. [9] and Peng Liu [16] propose architectures for intrusion tolerant database systems. However, these approaches are more focused on architectures for intrusion detection, and database recovery in case of an attack rather than proposing specific algorithms for performing the intrusion detection task on a DBMS. Shu et al. [24] also describe an architecture for securing web based database systems without

proposing any specific intrusion detection schemes. Lee et al. [14] describe a method for intrusion detection applicable only to real-time database systems. Among the most notable approaches towards a database-specific ID mechanism are those of Hu et al. [8] and DEMIDS [6]. Hu et al. provide mechanisms for finding dependency relationships among transactions and use this information to ascertain hidden anomalies in the database log. However, the work which is conceptually most similar to our work is DEMIDS. DEMIDS is a misuse detection system tailored to relational database systems. It uses audit-log data to derive profiles that describe typical users' activities. The drawback of such an approach, as mentioned earlier, is that the number of users for a database system can be quite large and maintaining/updating profiles for such large number of users is not a trivial task. Moreover, the approach used by DEMIDS to build user profiles assumes domain knowledge about the data structures and semantics encoded in a given database schema. This can adversely affect the general applicability of their methods. We, on the other hand, propose to build profiles using syntactic information from the SQL queries which makes our approach more generic than others.

Other approaches to database security include privacy-preserving DBMS. Architectures for hippocratic databases [3] have been proposed as a mechanism to preserve the privacy of data they manage. But, even though the architecture includes ID as a core component, it does not specify any methods for performing the ID task. Other recent work related with privacy-preserving databases mainly deals with topics such as data anonymization, privacy-preserving data mining, or fine-grained access control and auditing [15, 2, 20].

1.4. Paper road map

Next section describes the format of the log records and the three different representation levels we propose. Section 3 describes in detail the classifier used in our approach. Section 4 reports experimental results on both synthetic and real database traces. We conclude the paper by discussing future work.

2. Preliminaries

In order to identify user behavior, we use the database log files which are the primary source that naturally comes to mind when information regarding users' actions is needed. We use the log-file entries, after being processed, in order to form profiles of acceptable actions. Each entry of the log file is represented as a separate data unit and then these units are combined to form the desired profiles.

We assume that users of the database issue commands, where each command is a different entry of the log file, structured according to the SQL language. In particular, in the case of queries such commands have the format:

```
SELECT [DISTINCT] {TARGET-LIST}
FROM              {RELATION-LIST}
```

In order to build profiles, we need to transform the log file entries into a format that can be processed and analyzed. Therefore, we represent each entry by a data basic unit that contains three fields, and thus it is called *triplet*.

Triplets are our basic unit for viewing the log files and are the basic components for forming user and role profiles, since subjects' actions are characterized by sequences of such triplets. Each triplet contains information about the SQL command issued by the user, the set of relations accessed and the set of attributes within the relations that are referenced in the command. Therefore, the abstract form of such a command consists of three fields (*SQL Command, Relation Information, Attribute Information*). For sake of simplicity in the adopted notation, we represent a generic triplet with $T(c, \mathcal{R}, \mathcal{A})$ [1], where c corresponds to the command, $\mathcal{R}$ to the relation information and $\mathcal{A}$ to the attribute information.

Depending on the detail required in the profile-construction and in the ID phase, we generate the triplets from the log files entries according to three different strategies, each characterized by a different amount of recorded information.

The first strategy generates triplets recording the least amount of information. This strategy uses the so-called *coarse triplets* or *c-triplets* which only record counters on the number of relations and attributes required by a given command. Therefore, c-triplets only model the cardinality of the `TARGET-LIST` and `RELATION-LIST`, rather than the specific elements they contain. The c-triplets are defined as follows:

Definition 1. *A* **coarse triplet** *or* **c-triplet** *is a representation of a log record of the database log file. Each c-triplet consists of* 3 *fields (*`SQL-CMD, REL-COUNTER, ATTR-COUNTER`*). The first field is symbolic and corresponds to the issued SQL command, while the other two are numeric and correspond to the number of relations and attributes involved in the issued SQL command, respectively.*

In terms of the triplet notation $T()$, here both $\mathcal{R}$ and $\mathcal{A}$ have one element each, corresponding to the number of relations and attributes involved in the query respectively. Apparently, a large amount of valuable information in the log is ignored by the c-triples. It is however useful to consider

[1] Depending on the type of triplet the two arguments $\mathcal{R}$ and $\mathcal{A}$ can be of different types, but for simplicity and clarity we allow the symbols to be overloaded. Whenever the type of triplet is vital, we will explicitly specify it. However, when it is not specified our claims hold for all types of triplets. Additionally note that $\mathcal{R}$ and $\mathcal{A}$ can also be viewed generically as vectors.

such a "primitive" data unit since it is sufficient in the case of a small number of well-separated roles. Moreover, more sophisticated representations of log file entries are based on the definition of c-triplets.

The strategy records more information in the triplets record more information than the first one. Such a strategy uses the so-called *medium-grain triplets* or *m-triplets*. These triplets extend the coarse triplets by further exploiting the information in the query-log entry. Again, the m-triplet represents a single log entry of the database log file and we further consider each relation of the database separately by recording the number of attributes, accessed by the SQL command at hand, of each such relation. In terms of the triplet notation $T()$, $\mathcal{R}$ and $\mathcal{A}$ are vectors of the same size which is equal to the number of relations in the database. The m-triplets are defined as follows:

Definition 2. *A* **medium-grain triplet** *or* **m-triplet** *is a data object which corresponds to a single entry of the database log file and consists of* 3 *fields (*`SQL-CMD, REL-BIN[], ATTR-COUNTER[]`*). The first field is symbolic and corresponds to the issued SQL command, the second is a binary (bit) vector of size equal to the number of relations in the database. This bit vector contains 1 in its i-th position if the i-th relation is included in the SQL command. The third field of the triplet is a vector of size equal to the size of the* `REL-BIN[]` *vector. The i-th element of the* `ATTR-COUNTER[]` *vector corresponds to the number of attributes of the i-th relation that are involved in the SQL command.*

Finally, the third strategy is the one that extracts the largest amount of information from the log files. This strategy uses the so-called *fine triplets* or *f-triplets*; their structure is similar to that of the m-triplets. In particular, the first two fields of the f-triplets are the same of the m-triplets. F-triplets and m-triplets only differ for the third field which in the f-triplets is a vector, called `BIN-ATTR[[]]`, of vectors. The $i-$th element of `BIN-ATTR[[]]` is a vector corresponding to the i-th relation of the database and having size equal to the number of attributes of relation i. The vector elements are binary values indicating whether specific attributes of the relation i have been used by the SQL command.

Here, $\mathcal{R}$ is a vector of size equal to the number of relations in the database while $\mathcal{A}$ is a vector of the same size, but with each element i being a vector of size equal to the number of attributes in relation i. The formal definition of the f-triplets is as follows:

Definition 3. *A* **fine triplet** *or* **f-triplet** *is a detailed representation of a log entry. It consists of* 3 *fields (*`SQL-CMD, REL-BIN[], ATTR-BIN[[]]`*). The first field is symbolic and corresponds to the issued SQL command, the second is a binary vector that contains 1 in its i-th position if the i-th relation is included in the issued SQL command. The third field is a vector of N vectors, where N is the number of relations in the database. Element* `ATTR-BIN[i][j] = 1` *if the SQL command at hand accesses the j-th attribute of the i-th relation and* 0 *otherwise.*

Table 1 shows two SQL commands corresponding to select statements and their representations according to the three different types of triplet. In the example we consider a database consisting of two relations $R_1 = \{A1, B1, C1\}$ and $R_2 = \{B2, D2, E2\}$.

3. Classifier

This section describes in detail the classifier that has been used for forming the profiles as well as for deciding when to raise an intrusion alarm. Because information related to the roles of individuals is available from the database traces, the problem at hand is transformed into a classification (supervised learning) problem and has been addressed as such. For starters, we describe a standard method of solving the classification problem using the Naive Bayes Classifier. Despite some modeling assumptions that one would expect to degrade the performance of the classifier, Naive Bayes classifier has several properties that make it surprisingly useful in practice. Like all probabilistic classifiers under the Maximum Aposteriori Probability (MAP) decision rule, it arrives at the correct classification as long as the correct class is more probable than any other class; class probabilities do not have to be estimated very well. In other words, the overall classifier is robust to serious deficiencies of its underlying naive probability model [7]. Additionally, the probabilistic nature of the model enables us to raise an alarm when the probability of a user, acting according to the role he is claiming to have, is low.

In the sequel, we describe the general principles of the Naive Bayes classifier (for details see [18]) and then we show how they can be applied to our setting. In the supervised learning case each instance x of the data is described as a conjunction of attribute values, and the target function $f(x)$ can only take values from some finite set V. Apparently, the attributes correspond to the set of observations and the elements of V are the distinct classes observed. In the classification problem, a set of training examples D_T is provided, and a new instance with attribute values $(a_1, ..., a_n)$ is given. The goal is to predict the target value, or the class, of this new coming instance.

The approach we describe here is to assign to this new instance the most probable class value v_{MAP}, given the attributes $(a_1, ..., a_n)$ that describe it:

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j | a_1, a_2, ..., a_n).$$

SQL Command	c-triplet	m-triplet	f-triplet
SELECT A1, B1 FROM R1	select$< 1 >< 2 >$	select $< 1, 0 >< 2, 0 >$	select $< 1, 0 >$ $< [1, 1, 0, 0, 0], [0, 0, 0, 0, 0] >$
SELECT R1.A1, R1.C1, R2.B2, R2.D2 FROM R1, R2 WHERE R1.E1 = R2.E2	select$< 2 >< 4 >$	select $< 1, 1 >< 2, 2 >$	select $< 1, 1 >$ $< [1, 0, 1, 0, 0], [0, 1, 0, 1, 0] >$

Table 1. Triplets construction example

Using *Naive Bayes Theorem* we can rewrite the expression:

$$\begin{aligned} v_{MAP} &= \arg\max_{v_j \in V} P(v_j|a_1, a_2, ..., a_n) \\ &= \arg\max_{v_j \in V} \frac{P(a_1, a_2, ..., a_n|v_j)P(v_j)}{P(a_1, a_2, ..., a_n)} \\ &= \arg\max_{v_j \in V} P(a_1, a_2, ..., a_n|v_j)P(v_j). \end{aligned}$$

The last derivation is feasible because the denominator does not depend on the choice of v_j and thus it can be omitted from the $\arg\max$ argument. Estimating $p(v_j)$ is easy since it requires just counting its frequency in the training data. However calculating $P(a_1, a_2, ..., a_n|v_j)$ is not that easy considering a large dataset and a reasonably large number of attributes. The Naive Bayes classifier is based on the simplifying assumption that the attribute values are conditionally independent. In this case:

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j) \prod_i P(a_i|v_j) \qquad (1)$$

The computational cost is thus reduced significantly because calculating each one of the $P(a_i|v_j)$ requires a frequency counting over the tuples in the training data with class value equal to v_j.

The conditional independence assumption seems to solve the computational cost. However, there is another issue that needs to be discussed. Assume an event E occurring n_{Ej} number of times in the training dataset for a particular class C_j with size $|D_{Cj}|$. While the observed fraction ($\frac{n_{Ej}}{|D_{Cj}|}$) provides a good estimate of the probability in many cases, it provides poor estimates when n_{Ej} is very small. An obvious example is when $n_{Ej} = 0$. The corresponding zero probability will bias the classifier in an irreversible way, since according to the last equation the zero probability when multiplied with the other probability terms will give zero as its result. To avoid this difficulty we adopt a standard Bayesian approach in estimating this probability, using the m-estimate defined as follows:

Definition 4. *Given a dataset D_T with size $|D_T|$ and an event E that appears n_{Ej} times in the dataset for a class C_j with size $|D_{Cj}|$ and n_E times in the entire dataset, then the* m-estimate of the probability $p_e = \frac{n_{Ej}}{|D_{Cj}|}$ *is defined to be:*

$$p_E^m = \frac{n_{Ej} + m \cdot \frac{n_E}{|D_T|}}{|D_{Cj}| + m}.$$

The parameter m is a constant and is called equivalent sample size*, which determines how heavily to weight p_E relative to the observed data. If n_E is also* 0*, then we simply assume the probability $p_E^m = \frac{1}{|D_{Cj}|}$.*

So far we have described how the Naive Bayes Classifier works in a general setting. Applying the model to our intrusion detection framework is rather straightforward. The set of classes that we consider is the set of roles R in the system while the observations are the log-file triplets. In the sequel, we show how equation 1 applies when the three different types of triplets are considered.

For the case of c-triplets the application is straightforward since there are three attributes to consider namely the command, the relation information and the attribute information. Therefore, we have three numeric attributes c, R, A that correspond to those three fields. If $\mathcal{R}$ denotes the set of roles, predicting the role $r_j \in \mathcal{R}$ of a given observation (c_i, R_i, A_i) requires, in accordance to equation (1):

$$r_{\text{MAP}} = \arg\max_{r_j \in \mathcal{R}} P(r_j)P(c_i|r_j)P(R_i|r_j)P(A_i|r_j)$$

In the m-triplets, we again have three fields (c, R, A), where R and A are vectors of the same cardinality. Except for the command attribute c, the rest of the attributes considered in this case come from the product RA^T. Therefore there are $|R \cdot A^T| + 1$ attributes and equation (1) can be rewritten as follows:

$$r_{\text{MAP}} = \arg\max_{r_j \in \mathcal{R}} P(r_j) \prod_{i=1}^{N} p(R \cdot A^T[i]|r_j).$$

Finally, in the case of f-triplets, where fields R and A are vectors and vectors of vectors the corresponding equation is:

$$r_{\text{MAP}} = \arg\max_{r_j \in \mathcal{R}} P(r_j) \prod_{i=1}^{N} p(R[i] \cdot A[i]|r_j).$$

Now with the above definitions in place, the ID task is rather straightforward. For every new query, its r_{MAP} is predicted by the trained classifier. If this r_{MAP} is different from the original role associated with the query, an alarm is raised.

The procedure for ID can easily be generalized for the case when a user can exercise more than one role at a time. This is because our method detects intruders on a per query basis and not on a per user basis; hence, as long as the role associated with the query is consistent with the role predicted by the classifier, the system will not raise an alarm.

4. Experimental evaluation

In this section, we report results from our experimental evaluation of the proposed approach and illustrate its performance as an ID mechanism. Our experimental setting consists of experiments with both synthetic and real data sets. The objective of this evaluation is two-fold. First, we present results comparing the behavior of our classifier when log files are modeled using the three different triplet representations. Second, we measure the performance of our classifier in terms of the computational cost associated with the training and detection phases. Before describing our experimental findings, we give a brief outline of the data sets and the quality measures we use for our evaluation.

4.1. Quality Measures

For the experimental evaluation, we analyze the quality of results of our approach as an ID mechanism using the standard measures of Precision and Recall. The precision and recall statistics are defined as follows:

$$Precision = \frac{\text{\# True Positives}}{\text{\# True Positives + \# False Positives}}$$

$$Recall = \frac{\text{\# True Positives}}{\text{\# True Positives + \# False Negatives}}$$

Here, # False Positives is the number of false alarms while # False Negatives is the number of times the system is not able to detect the anomalous queries.

4.2. Data sets

Synthetic data sets: The synthetic data are generated according to the following model: Each role r has a probability, $p(r)$, of appearing in the log file. Additionally, for each role r the generating model specifies the following three probabilities: (i) the probability of using a command cmd given the role, $p(\text{cmd}|r)$, (ii) the probability of accessing a table T given the role, $p(T|r)$ and (iii) the probability of accessing an attribute within a table $a \in T$ given the role and the table $p(a|r, T)$.

Real Data set: The real data set used for evaluating our approach consists of over 6000 SQL traces from eight different applications submitting queries to a MS SQL server database. The database itself consists of 119 tables with 1142 attributes in all. For more detailed description of the data set we refer the reader to [25].

Intruder Queries generation: The intruder/anomalous queries are generated by taking into account the insider threat scenario. They are taken from the same probability distribution as of normal queries, but with role information negated. For example, if the role information associated with a normal query is 0, then we simply change the role to any role other than 0 to make the query anomalous.

4.3. Results

Test Cases 1, 2 and 3 show the relative accuracy of our classifier with respect to the three triplet representations. Test Case 1 (Figure 2) shows the inferior performance of c-triplets compared to the other two alternatives. In this test case, each role issues queries accessing approximately an equal number of columns but from different tables. For this reason, the queries when modeled by the c-triplets show a lot of homogeneity across the roles. Hence, the classifier is not effective in distinguishing between queries across different roles. This results in the low precision and recall values for c-triplets. Test Case 2 (Figure 3) establishes the superiority of f-triplets over c and m-triplets. In this case, there is a partial overlap in the table access pattern of the queries issued by the various roles. In addition to this, each role accesses only a subset of the columns within a table with a high probability. In such a scenario, f-triplets perform the best because they take into account the column access pattern of the queries within a table, unlike m-triplets which only take into account the number of columns accessed per table and c-triplets which only maintain a count of total number of columns and tables accessed. Also, the performance of all three triplet types improves with increasing amount of training data. Test Case 3 (Figure 4) is a variant of Test Case 2 with varying number of roles and constant training data (1000 tuples). As expected, f-triplets give the best results in this case as well.

Finally, we tested our classifier on the real data set. For this experiment, we consider 6602 SQL traces from 4 different applications as the training data for the classifier. Tables 2 and 3 show the quality of the results for the three different triplet types. Not surprisingly, high quality is achieved for all triplet types.

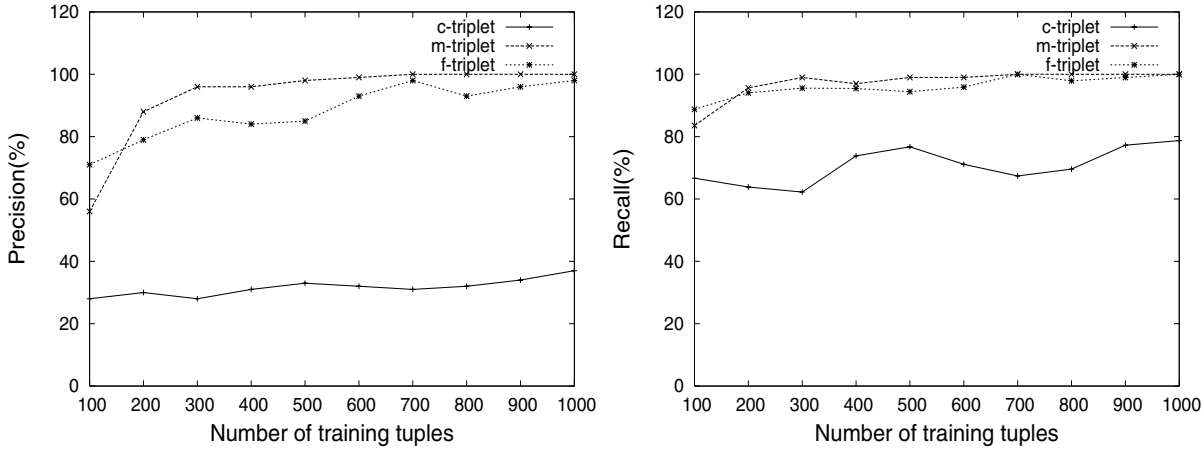

Figure 2. Test Case 1: Precision and Recall statistics

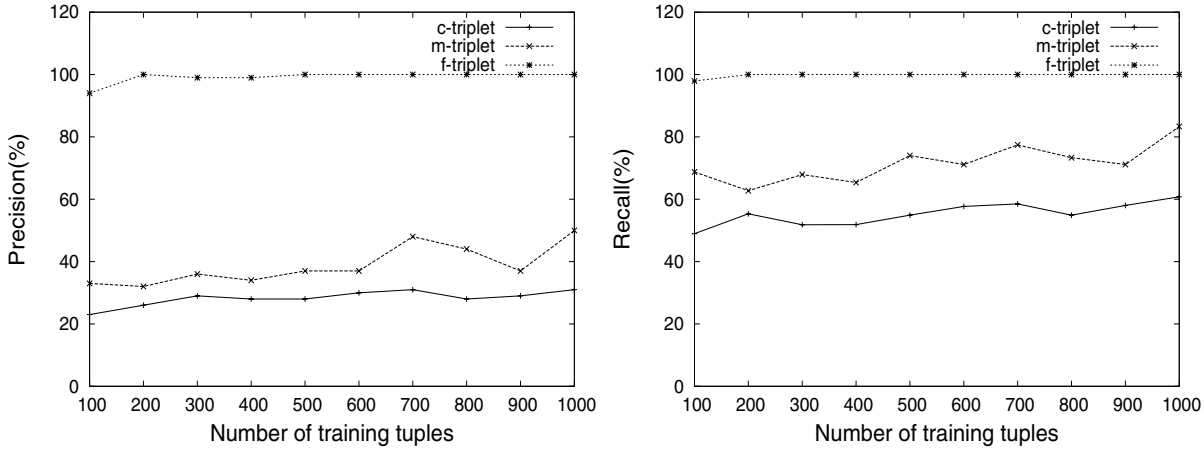

Figure 3. Test Case 2: Precision and Recall statistics

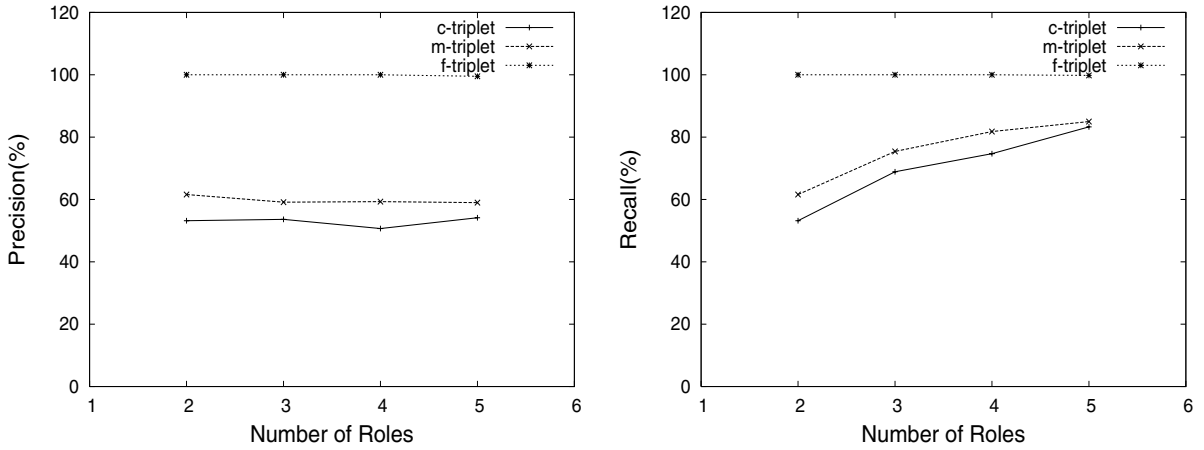

Figure 4. Test Case 3: Precision and Recall statistics

Overall, the experimental evaluation reveals that in most cases m and f-triplets capture the behavior of the data much better than the c-triplets.

Triplet type	False Negatives(%)	False Positives(%)
c-triplet	2	18
m-triplet	2	15
f-triplet	3.4	16

Table 2. Real data, False negatives/positives statistics

Triplet type	Recall(%)	Precision(%)
c-triplet	82	97.62
m-triplet	85	97.7
f-triplet	84	96.1

Table 3. Real data, Precision and Recall statistics

4.4. Performance

One of the many desirable properties of Naive Bayes Classifier is its low computational cost. This is due to the simple underlying probability model that assumes conditional independence among the attributes. Due to this, the role profiles can be built by the classifier in just one pass over the training data. In this section we present results demonstrating the low execution time of our approach, for both the training phase and the intrusion detection phase.

Figure 5 shows the training time as a function of the number of tuples in the training data, for the three proposed triplet representations. As expected, the training time increases linearly with the number of tuples in the training data. Moreover, the training time for c-triplets is negligible, since c-triplets need just three attributes to be represented. One notable observation is that the training time for both m and f-triplets is of the same order of magnitude. This is because both m and f-triplet representations contain the same number of attributes. Still, the time for f-triplets is higher than m-triplets approximately by a factor of 3. The reason for this difference is the vectors of vectors representation of f-triplets which makes their attribute-domain values much larger than that of m-triplets. Figure 6 shows the training time as a function of the number of attributes in the underlying database tables. Not surprisingly, the training time increases with the number of attributes for both m and f-triplets. Again, the order of magnitude of training time is same for both of them.

The time complexity of the detection algorithm for our classifier is in $O(R \times A)$ where R is the number of roles present in the database and A is the number of attributes considered by the classifier. Figure 7 gives the variation of the detection time per query for databases with different total number of attributes. As expected, the detection time increases almost linearly for increasing values in the number of attributes for both m and f-triplet types. However, the worst-case scenario is still 1.54 milliseconds per query (f-triplets, 100 attributes), which is negligible.

Overall, the performance experiments confirm the low overhead associated with our approach. This gives us an opportunity to explore the possibility of integrating our approach with other query processing features of a database for an integrated online ID mechanism embedded inside a database.

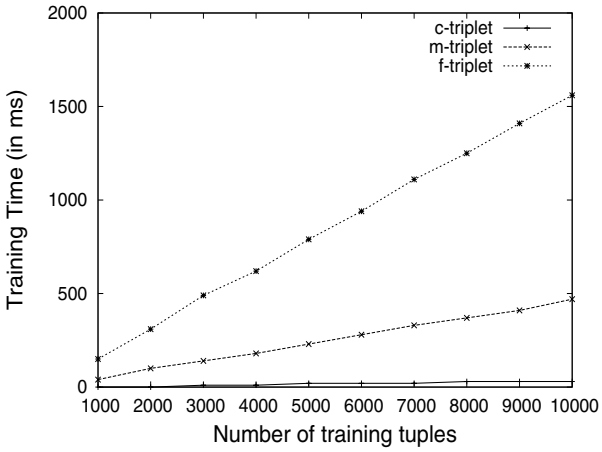

Figure 5. Training Time vs Training Data

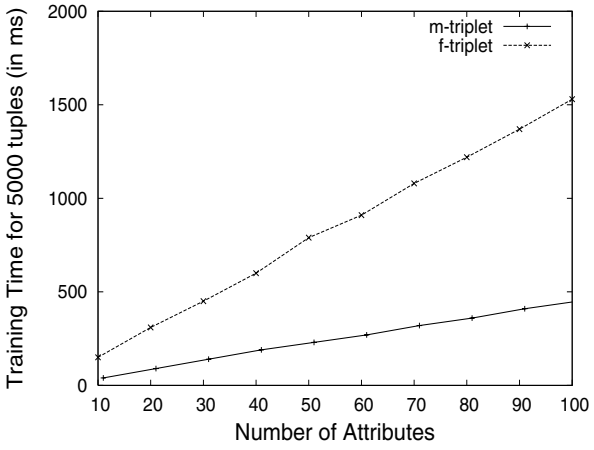

Figure 6. Training Time vs Number of Attributes

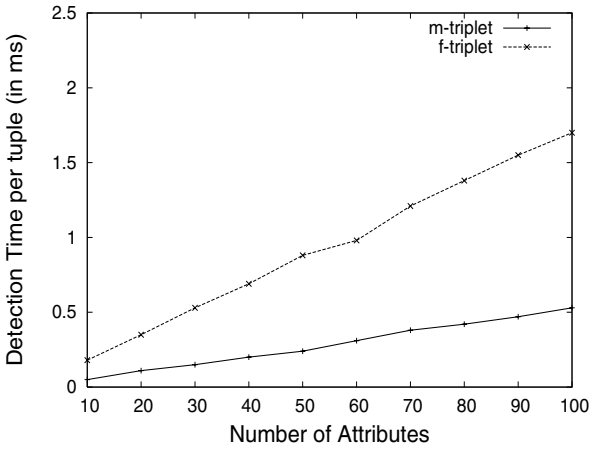

Figure 7. Detection Time vs Number of Attributes

5. Conclusions and Future Work

In this paper we investigated incorporating an intrusion detection mechanism inside a DBMS. We considered three models, of different granularity, to represent the log records appearing in the database log files. In that way, we managed to extract useful information from the log records regarding the access patterns of the users. Since role information was available in the log records, we used it for training a classifier that was then used as the basic component for our intrusion detection mechanism. Experimental results for both real and synthetic data sets showed that our methods perform reasonably well.

As part of future work, we will investigate the case when role information is not present in the log records. The problem of forming user profiles is then clearly unsupervised and thus can be treated similarly to a clustering problem. In this case, standard clustering algorithms can be employed for constructing groups of users that seem to behave similarly. These groups may also help the Database Administrator (DBA) to decide which roles to define. The intrusion detection phase can then be addressed as an outlier detection problem. Another direction for future research is to maintain sub-profiles within a role profile to capture the normal user behavior in a more intuitive sense. For example, consider the behavior of a reservation agent that needs to add bookings, modify bookings, cancel bookings, forward bookings, run statistics on bookings, etc..Each of these can be a separate class of behavior within the profile of a reservation agent role. The intrusion detection task can then be carried out as a combination of supervised and anomaly detection approaches. In addition to this, we will also explore better representations of SQL queries so as to capture not only the syntactic but also the semantic information contained in them.

6. Acknowledgments

The authors would like to thank the anonymous referees for the many invaluable suggestions that lead to a much improved version of this paper.

References

[1] K. H. A. Hoglund and A. Sorvari. A computer host-based user anomaly detection using the self-organizing map. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)*, 2000.

[2] R. Agrawal, R. J. B. Jr., C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance with a hippocratic database. In *Proceedings of the 30th international conference on Very Large Data Bases (VLDB)*, pages 516–527, 2004.

[3] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proceedings of the 28th international conference on Very Large Data Bases (VLDB)*, pages 143–154. Morgan-Kaufmann, 2002.

[4] A. Anton, E.Bertino, N.Li, and T.Yu. A roadmap for comprehensive online privacy policies. In *CERIAS Technical Report, 2004-47*, 2004.

[5] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Chalmers Univ., Mar. 2000.

[6] C. Chung, M. Gertz, and K. Levitt. Demids: a misuse detection system for database systems. In *Proceedings of Integrity and Internal Control in Information Systems: Strategic Views on the Need for Control. IFIP TC11 WG11.5 Third Working Conference*, 2000.

[7] P. Domingos and M. J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[8] Y. Hu and B. Panda. Identification of malicious transactions in database systems. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, 2003.

[9] Z. Jian-ming and M. Jiang-feng. Intrusion-tolerant based architecture for database system security. *Journal of Xidian University*, 3(1), February 2003.

[10] J. B. Joshi, R. Bhatti, E. Bertino, and A. Ghafoor. Access-control language for multidomain environments. *IEEE Internet Computing*, 8(6):40–50, 2004.

[11] J.Vaidya and C. Clifton. Privacy-preserving data mining: Why, how, and when. *IEEE Security and Privacy*, 2(6):19–27, 2004.

[12] G. Karjoth. Access control with ibm tivoli access manager. *ACM Transactions on Information and Systems Security (TISSEC)*, 6(2):232–257, 2003.

[13] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and Systems Security (TISSEC)*, 2(3):295–331, 1999.

[14] V. Lee, J. Stankovic, and S. Son. Intrusion detection in real-time databases via time signatures. In *Proceedings of the Sixth IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2000.

[15] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. J. DeWitt. Limiting disclosure in hippocratic databases. In *Proceedings of the 30th international conference on Very Large Data Bases (VLDB)*, pages 108–119, 2004.

[16] P. Liu. Architectures for intrusion tolerant database systems. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, 2002.

[17] T. Lunt, A. Tamaru, F. Gilham, R. Jagannathan, P. Neumann, H. Javitz, A. Valdes, and T. Garvey. A real - time intrusion detection expert system (ides) - final technical report. *Technical Report, Computer Science Laboratory, SRI International*, 1992.

[18] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[19] R. Sandhu, D. Ferraiolo, and R. Kuhn. The nist model for role based access control: Towards a unified standard. In *Proceedings of the 5th ACM Workshop on Role Based Access Control*, 2000.

[20] R. Shariq, M. Alberto, S. S., and R. Prasan. Extending query rewriting techniques for fine-grained access control. In *Proceedings of ACM SIGMOD, International Conference on Management of Data*, 2004.

[21] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):571–588, 2002.

[22] R. Talpade, G. Kim, and S. Khurana. Nomad: Traffic-based network monitoring framework for anomaly detection. In *Proceedings of the 4th IEEE Symposium on Computers and Communications*, 1999.

[23] V. S. Verykios, E. Bertino, I.Nai-Fovino, L. P. Provenza, Y. Saygin, and Y.Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Record*, 33(1):50–57, 2004.

[24] S. Wenhui and T. Tan. A novel intrusion detection system model for securing web-based database systems. In *Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC)*, 2001.

[25] Q. Yao, A. An, and X. Huang. Finding and analyzing database user sessions. In *Proceedings of the 10th International Conference on Database Systems for Advanced Applications (DASFAA)*, 2005.