Managing Risks in RBAC Employed Distributed Environments

Ebru Celikel[1], Murat Kantarcioglu[1], Bhavani Thuraisingham[1], and Elisa Bertino[2]

[1] University of Texas at Dallas Richardson, TX 75083 USA
[2] Purdue University West Lafayette, IN 47907 USA
{ebru.celikel,muratk,bhavani.thuraisingham}@utdallas.edu,
bertino@cs.purdue.edu

Abstract. Role Based Access Control (RBAC) has been introduced in an effort to facilitate authorization in database systems. It introduces roles as a new layer in between users and permissions. This not only provides a well maintained access granting mechanism, but also alleviates the burden to manage multiple users. While providing comprehensive access control, current RBAC models and systems do not take into consideration the possible risks that can be incurred with role misuse. In distributed environments a large number of users are a very common case, and a considerable number of them are first time users. This fact magnifies the need to measure risk before and after granting an access. We investigate the means of managing risks in RBAC employed distributed environments and introduce a probability based novel risk model. Based on each role, we use information about user credentials, current user queries, role history log and expected utility to calculate the overall risk. By executing data mining on query logs, our scheme generates normal query clusters. It then assigns different risk levels to individual queries, depending on how far they are from the normal clusters. We employ three types of granularity to represent queries in our architecture. We present experimental results on real data sets and compare the performances of the three granularity levels.

Keywords: RBAC, security, access control, risk modeling, data mining.

1 Introduction

Today more than ever, data sharing among a variety of users from different domains and environments is a key requirement. Data sharing is crucial for decision making processes in that it enables individuals to take decisions based on complete and accurate information. Data sharing, however, has to be carried out by safeguarding data confidentiality through the use of an access control mechanism. To provide adequate access control, database systems thus necessitate a security tool combining together policies, technologies and people [18]. Unfortunately, security requirements of a database are usually contradictory to the user requirements: On one hand security forces us to have strict limitations over permissions; on the other hand, users demand more permission to accomplish their tasks [16]. Furthermore, in a typical distributed environment, users tend to establish coalitions for data sharing purposes. Such an

R. Meersman and Z. Tari et al. (Eds.): OTM 2007, Part II, LNCS 4804, pp. 1548–1566, 2007.

environment is typically not closed and its users are very often at different locations. Moreover, access control must not affect the performance of the query processing engine, security [20] and other components of the database system.

Role Based Access Control (RBAC) model [11, 15], is a practical solution to the problems listed above. The introduction of roles as an intermediate level between users and permissions makes user management easier. The use of roles in RBAC also allows one to determine who can take what actions on which data [12]. In real world, we expect role and permission associations to change less frequently than user and permission associations. This is because, organizations usually have a well defined set of privileges for each role and they stay stable; whereas users can change positions, hence require dynamic allocation of permissions. By its ability to predefine role and permission relationships, RBAC supports the three fundamental security principles as the least privilege, separation of duties and data abstraction [26]. All these features make RBAC feasible and easy to use.

In RBAC model, credentials are used to determine legitimate users and thereafter users are assigned to roles. But RBAC does not consider the risk in this process. When we look at the potential sources of risk in an RBAC administered database, we see that mainly two sources of risk contribute to the overall risk evaluation: one is the inherent risk that is incurred by user credentials such as the location of connection, if the user is the first time user or not, etc., and the other is the risk resulting by role misuse or abuse. By role misuse we refer to the unintentional incorrect use of a role, whereas by role abuse we refer to the intentional incorrect use. For the sake of simplicity, we denote both intentional and unintentional cases with the same phrase as role misuse throughout the paper. Given user credentials, RBAC perfectly handles the risk incurred by credentials: It eliminates the illegal access attempts by totally rejecting them. Likewise, in case of access requests exceeding the actual role definitions, RBAC rejects these attempts. Still, there will be users attempting to exploit their already assigned permissions by using them over and over again. Unfortunately, RBAC does not consider this type of role misuse. So, in that sense every access attempt carries a potential risk.

In this paper, we address the security of RBAC employed distributed databases by focusing on the risk management in such systems. Motivated by the strong and flexible access control facility that RBAC provides, we introduce an extension to it. We design and implement a mathematical model to measure risk, so that RBAC provides improved security for access control. We know that several factors such as immature and improper enforcement of constraints, delegation processes and/or role hierarchy construction contribute to the risk in databases. We assume that these factors are all mitigated with comprehensive risk management and we only focus on risks caused by user credentials and misuse.

As a motivating example, assume that several companies from various countries come together under an international organization for business purposes. Their aim is to combine their resources to conduct business all around the world. The reason for that is two fold: First, companies cannot realize projects individually with limited resources in their own countries, and second sometimes it is economically more feasible to make an investment with partners in another country. The resources each country has are different: for instance some countries have very fertile soil for good plantation, some have money, others are good at technology and equipment, etc. To

conduct joint projects, two or more of the member countries initiate a coalition. During the lifetime of a coalition, the participating countries need to establish and maintain mutual trust for each other. Imagine that countries A, B and C come together to start up a new factory in country C. Let's call this coalition as coalition ABC. As long as this coalition is active, the participating companies from countries A, B and C will be exchanging information about several topics as the amount of money they will invest for the new factory, the details about particular resources each country will provide, the physical location of the factory to be settled in country C, etc. Even some other countries, say country D that is not a participating country in the coalition ABC, but a member of the organization, may request information from participants about this coalition. At this point, countries A, B and C may not trust each other completely. But again they need to communicate and it is very important for them to keep their project secret, so that no other country steals the idea before they start the new factory. While exchanging information, countries A, B, C –and also country D in case it communicates with the coalition- require a secure access control mechanism to identify users who would attempt to misuse the permissions granted to them by their role definitions. For example, assume that a human resources personnel from country A has permission to ask salary information for employees and he asks these questions: "What is the salary of the general manager of the new factory?", "What is the salary of the account manager of the new factory?", "What is the salary of human resources manager of the new factory?" etc. to reveal the salaries of the whole employers in the new factory. Even if he is a legal user, submitting multiple salary questions to the system should indicate a suspicious situation. In that sense, every communication in this business coalition incurs a potential risk.

Assume that RBAC is employed to detect unauthorized access attempts, and authorized but still illegal requests that exceed the actual permissions in this sample database. While facilitating access control in multiple aspects, RBAC remains inadequate in detecting the potential risk of users' misuse. To improve the strength of RBAC, we propose a quantitative model to evaluate risk in such a database. Throughout the paper, we use the international business organization example for further reference.

1.1 Our Contribution

RBAC is an effective tool to protect information assets from internal and external threats [18]. It gets user credentials to assign legal users to roles. While doing that, RBAC provides flawless control over users in two ways: It totally rejects users having credentials that do not comply with the role requirements, as well as user attempts that ask for more than what their role actually allows. Yet, employing RBAC alone is not enough to eliminate security threats. Even if roles are well defined, every access request carries a potential risk of role misuse. To provide a comprehensive security, we need to analyze queries to measure the risk that is incurred to the system by their submission, and behave accordingly.

In this study, we address the security requirement of an RBAC administered distributed environment. Our aim is to extend the strength of this standard access control mechanism by embedding a mathematical risk evaluation model in it. We propose a quantitative approach to assess risk. The risk model we put forward is novel

in the sense that, it dynamically measures the level of risk in granting an access request. The structure of RBAC model allows us the flexibility to place our risk evaluation scheme either in the middle of user to role assignments or in the middle of role to permission assignments.

Mainly addressing the issue of security, our design introduces a risk adaptive access control mechanism (RAdAC) [13]. Several risk factors contribute to the calculation of risk in our design. We list these factors as user credentials, current user queries together with old queries and the utility expected by executing the query. Obviously, not every risk factor should have an equal share in the overall risk calculation. Hence, we assign different weight indices to each factor, depending on how important it is in the overall risk evaluation. At the end, our system sums the weighted contribution of each factor to yield a single risk value.

While measuring risk, precision is very important. In order to obtain better precision, we incorporate data mining on the set of current queries and the role history log. For that, we implement anomaly (outlier) detection by using K-means clustering, which is an unsupervised classification algorithm to generate query clusters. We then analyze individual queries to determine how far a single query is from already formed clusters. Afterwards, we assign a risk value to each query, where the value assigned is linearly proportional to its distance from the nearest cluster. This, in turn, forms the risk value for the role history factor of the whole risk evaluation scheme.

The rest of the paper is organized as follows: Section 2 gives background information about risk evaluation in distributed environments. Section 3 describes our risk measurement model in detail. Section 4 presents implementation details together with experimental results. The last section is Section 5, where conclusion and future work are presented.

2 Background

The In literature, various definitions of risk exist. Economists define it as a special type of uncertainty involving a variation from the expected outcome. They measure risk with the standard deviation of all probable outcomes [2]. From the computational point of view, risk is defined as a combination of likelihood and impact of an event. Trust is a tightly coupled concept with risk: a system with high risk has a low level of trust and vice versa. This indicates a tradeoff between risk and trust and these terms are sometimes used interchangeably.

Several studies for trust evaluation have been conducted. The Secure Environments for Collaboration among Ubiquitous Roaming Entities (SECURE) project [3, 5, 6, 7, 8, 9, 10, 17] is one of them. SECURE is a result of a comprehensive and ongoing work. With the tool they develop in the SECURE project, the authors try to form a general basis for trust and risk reasoning, as well as a security policy framework to be embedded into various applications. Regarding the above mentioned definition of risk, they base risk on two principals: One being the other principal's trustworthiness (likelihood) and the other being outcome's cost (impact), which can either be in the form of a benefit or loss. Their system represents the cost distribution as a cost-Probability Density Function (PDF). SECURE is made up of three components: a risk evaluator, whose task is to assess the possible cost-PDFs using the trust information

generated by the trust calculator; a trust calculator which determines the likelihood of risk by considering the principal's identity and other parameters of the action; and a request analyzer which combines the cost-PDFs of each outcome to determine whether the action will be taken or not.

English et al. propose an extension to the SECURE project [English et al, 2003]. Forming the premises for risk assessment and interaction/collaboration decisions, their architecture dynamically analyzes trust in three levels as formation, evolution and exploitation. The sources of trust in their system are observations, recommend-dations and reputation. They add a collaboration monitoring and evaluation that involves a feedback mechanism to the end of the decision making process.

Another trust based study has been developed by Xiong and Liu [25]. Based on reputation, they develop a tool called PeerTrust for evaluating and comparing the trustworthiness of entities in a peer-to-peer decentralized network. Their approach is motivated by the idea that the trust models relying solely on other peers' feedback is inadequate. For that, the authors add three factors to trust computation: (1) The amount of satisfaction, (2) the number of interactions and (3) the balance factor of trust, which is used to neutralize the potential of false feedback of peers.

Abdul-Rahman and Hailes define a trust model derived from the sociological characteristics of trust [1]. They represent trust as a combination of experience (denoted by direct trust) and reputation (denoted as recommender trust). While direct trust relies on the agent whose trustworthiness is evaluated, the context and the degree of trustworthiness; recommender trust is based on another agent, context and the degree of trustworthiness.

All of the aforementioned models merely base their work on trust, which is calculated by using other principals' recommendations and system outcomes only. To the best of our knowledge, what is missing in prior research is an actual quantitative risk evaluation. Moreover, some of the models are implemented in non-RBAC administered environments. We propose a risk measurement model to fill this gap: our study introduces the notion of dynamic and adaptive risk measurement in RBAC employed distributed environments. As part of our work, we employ data mining to detect anomalies, i.e. queries with higher risk.

Similar to our approach, Bertino et al. use data mining on RBAC administered databases. In [4], they employ data mining to detect intrusions. They use the Naïve Bayes algorithm, which is a supervised learning technique to classify queries as intrusions or not. On the other hand, we use an unsupervised learning algorithm (K-means clustering) to detect outlier queries in our work. Our goal is not intrusion detection.

Data mining algorithms have several other applications in the field of RBAC. Schlegelmich and Steffens's study, where they introduce a role mining tool with a new approach, is an example of this [16]. Another implementation belongs to Vaidya et al. [21]. Their work introduces RoleMiner, an unsupervised role mining tool.

3 Our Risk Evaluation Scheme

In this paper, we address the risk management problem in an RBAC employed database and propose a mathematical model for measuring risk in such environments.

Since the amount of risk involved in granting an access request may depend on various reasons, we base our quantitative risk calculation on several risk factors. With respect to the fact that the management of RBAC is very flexible; users may be dynamically added to roles, even after permissions have been granted to roles. As a basis for our work, we consider a simple RBAC scenario where roles are assigned to users, permissions are assigned to roles and after that actual execution of transactions begin [11]. These relationships in the sample scenario are schematized in Figure 1.

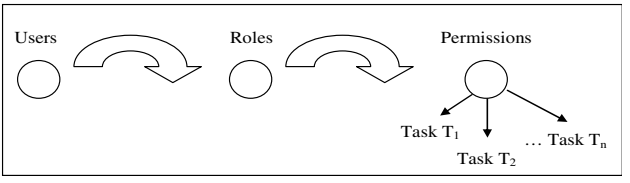

Fig. 1. A Sample Scenario in RBAC Model

In our work, we take advantage of this simple fact of RBAC: Users assigned to the same role are expected to behave similarly. This is because roles are already granted access to perform a predefined set of actions and users are supposed to adopt these roles. As long as users obey their role requirements, i.e. submit queries that are in accordance with their current roles, we simply assign reasonable risk levels to them. But when they behave in a manner that is contradictory to their role definitions, we detect this as a role misuse. In this case we label this behavior as an anomaly and assign high risk level to the current owner of this role.

The problem of risk assessment for a database user is analogous to that of a potential customer who makes a credit card application to a bank. Just as the bank asks the prospective customer's personal information before releasing a card, our system gets user credentials for identification purposes before granting the user's access request. After getting personal information, the first thing the bank does is to search its history logs to find out previous records for the prospective customer. At this point, an important difference between this example and our work needs to be pointed: In the bank example, a global credit history is used to keep track of the customer histories. On the other hand, in our design, we use local log files to store user histories. Going back to the bank example, if record(s) with previous transactions for the prospective customer are found, then the bank investigates whether or not the customer well behaved (made credit payments on time) before. If no such records are found, then the bank refers to the statistics of similar applications made before and tries to find out how many of the brand new customers recorded good credit histories. Other than these, the bank can take its decision only on the user's personal information, i.e. his credentials. At this point, the bank does one of the following: (1) It takes the risk and gives the customer the credit card, because it needs customers and money. (2) It simply refuses the application by just saying that he has insufficient credit history.

Upon receiving an access request to the database, our risk model behaves like the bank: it first retrieves queries that are considered to be normal and categorizes them. The normal queries in our system are the ones that have been submitted to the

database before and have been granted. Then we get individual queries submitted by the current user and detect how far each of them is from the normal queries. We expect that users with the same role definitions behave similarly. If the individual query is not close enough to any of the normal role behaviors, we assign a high level of risk for this particular query of the current user. If the individual query is close to any of the normal role behaviors, then we assign a reasonable (low) risk level to him. By repeating the same risk assignment for each query of the user, we end up with an average overall risk value for that particular user. In case this risk level is too high, we most probably reject his access request. There is another option as immediately rejecting the user access, once a query submitted by him is detected to have too high risk.

We design our risk evaluation mechanism such that, it can be embedded into the RBAC model. In the sample scenario given in Figure 1, our design can find a place to itself either in between user to role assignments, or in between role to permission assignments. In the former we measure the amount of risk involved in assigning a user to a particular role, and in the latter we measure the amount of risk in granting the access rights (permissions) to the pre-defined roles. For both cases, the implementation of our design does not change. The only thing that changes is the input and output to the system.

We give a diagram of our design in Figure 2. In this design, we assume that our risk evaluation mechanism is placed into role to permission assignment phase of the RBAC model. As Figure 2 shows, there are four risk factors contributing to the overall risk calculation: User credentials, set of current user queries, role history logs, and the amount of utility expected by the execution of queries.

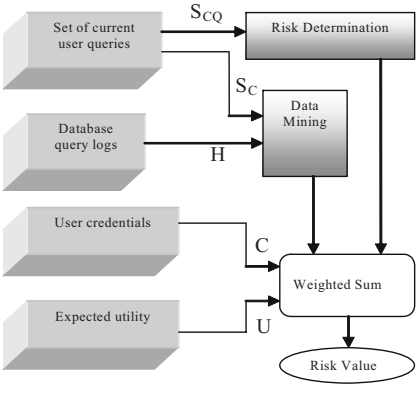

Fig. 2. A Sample Scenario in RBAC Model

To assign users to roles in a distributed environment, we need to collect user information, i.e. credentials to differentiate several users from each other. The other reason why we need credentials is to localize the place of connection. We collect each user's credentials containing the information as user name, IP number of the connection, date and time he last connected, etc. In our design, we denote user credentials with C.

In order to determine the amount of risk a user carries, collecting his credentials is not enough. We further need to analyze the queries submitted by him. To fairly calculate the value of risk, we make our calculation over a set, other than individual queries (Q). We refer to the collection of queries submitted by the same user as the set of current user queries (denoted by S_{CQ}). This is actually the history log for the current user. While handling the set of current user queries, we employ a sliding window mechanism. In practice, we can set different values for the window size. We repeated tests with different window sizes as 10, 20 and 30. Since the results we obtained did not change significantly, we set the smallest window size, i.e. 10 for our sample implementation. The size of the sliding window can easily be adjusted to observe our system's response to changing window sizes.

The contents of S_{CQ} are in the form of SQL queries. In our implementation, we process the set of user queries twice: (1) to derive the nature of the query, which is obtained by tokenizing each query into the SQL command itself, relation name(s) and attribute name(s); (2) to compare it with the history of other user queries, i.e. role history.

The third risk factor in our architecture is the role history log, which is in the same format as the set of current user queries. This is a large file containing several user queries and we denote it by H. While measuring risk, we use this simple idea: the best way to estimate the future is to observe the past. So, before assigning a risk value to the current user, we observe the database logs. This observation gives us an insight about how the current user's behavior will be like in the future.

The last component that we use for evaluating risk is the amount of expected utility assuming that the access request is granted. We denote the expected utility by U.

So, given the set of risk factors, where C is the user credentials, S_{CQ} is the set of current user queries (with Q denoting the individual elements of the set S_{CQ}), H is the role history log and U is the expected utility, we suggest that the total expected return (E) would be an accurate indication of risk. We also propose representing the total expected return with a statistical utility function that we give in Equation 1:

$$E = (1 - P) \times U - P \times M_Q \tag{1}$$

In the above equation, M_Q is the estimated misuse cost that is incurred to the system by each individual query (Q) of the set of current queries (S_{CQ}). This cost is the inherent cost that the system incurs depending from the type of the SQL query (e.g. SELECT, UPDATE, ADD, DELETE, etc.), the relation(s) (e.g. SALARY table, HOBBIES table, etc.), and the attribute(s) (e.g. SSN attribute, DATEOFBIRTH attribute, etc.). Moreover, in the above equation P denotes the probability of role misuse; its estimation is expressed by the conditional probability given in Equation 2:

$$P = Pr(Misuse \mid C, Q, S_{CQ}, H) \tag{2}$$

In the following subsections, we give a detailed explanation of each risk factor that we use in our model.

3.1 User Credentials (C)

In a distributed environment, where the number of users is usually very large, credentials are the essential elements to identify and differentiate users. User

credential(s) is the first risk factor that directly contributes to the calculation of the probability of role misuse (Equation 2). In the risk model we propose, user credentials are the identification components issued by participants in the database. We utilize user credentials to obtain user's personal information, together with his IP address, and the information whether he has made an access request before or not. In our international business organization example, the user credentials are the company names that contribute to the coalition, and their country of origin. We may further request information about whether or not a request owner participated in a joint project before. Because, the existence of a former relationship can help us set the level of initial risk with higher precision. As for the current implementation, we assume that user credentials are input to the system via a secure and complete means: e.g. smart card, RFID, automated user entry, etc.

3.2 Set of Current User Queries (S_{CQ})

The second risk factor in the calculation of the probability of role misuse (Equation 2) is the set of current user queries, S_{CQ}. This is actually the history log for the current user. In our system, the decision whether to grant or deny an access request relies on the estimated misuse cost (M_Q) of each query (Q). This is what we call the nature of the query. In order to determine the nature of the query, we do the following: By analyzing a tokenized representation of the query, we determine the type of the query, i.e. whether an insertion, deletion, or modification and what critical relation(s) and/or attribute(s) it attempts to access. When we assign a risk value to the current query, the first step we execute is to check whether this user has submitted queries to the system before, i.e. we check the current user's history logs. For such purpose, we collect queries submitted by the same user under a group; we call the set of current user queries (S_{CQ}) and treat it using a sliding window mechanism. Each time we calculate the risk value for the current query, we move the sliding window to process the new query for the same user. In case the user has no history yet, i.e. he is a first time user, then S_{CQ} contains a single query. So, as an unknown user, we assign the highest level of initial risk to his query. Eventually, if he submits new queries, we update his query risk level as he builds up his own history in our system. This update mechanism is what makes our risk calculation scheme dynamic and adaptive.

For the ongoing international organization example, the probable set of current user queries can include the following questions: "How much money does country A/B put in the joint project?", "What is the product that we will produce in the new factory?", "In which city in country C we will set up our new factory?", "What is the date we will start functioning the factory?", etc.

The order of processing for S_{CQ} is after we get the user's credentials and already assigned a credentials risk to him. Even if the user has low credentials risk, S_{CQ} may still contain one or more queries that should be detected as highly risky. For example, a user from country A (so, supposedly a legitimate user and hence carries low risk in terms of credentials) may repeatedly ask questions as "What is the name and SSN of the general manager for the new factory?", "What is the name and SSN of the account manager for the new factory?", "What is the name and SSN of the director of the human resources department for the new factory?", etc. These insistent queries to reveal the identity of employees should attract suspicion and our risk model assigns high risk to the owner of such queries.

In order to compute the overall risk value for S_{CQ}, we need to assign an estimated misuse cost M_Q (Equation 1) to each individual query Q of S_{CQ}. We could assume that M_Q is a fixed value input to the system with each query submission. Instead, we provide a better estimation that will determine M_Q in an automated and more precise manner. For that, we assign predetermined weight indices to each SQL command, to each relation, as well as to each attribute in the database. The decision to assign which weight to each attribute and each relation is totally domain specific. In general, we deliberately assign higher weights for critical attributes and tables. For example, a patient relation is more critical than hospital facilities relation. So, the attributes in the former are assigned higher weights than that of the latter. Afterwards, we tokenize each query to separate SQL commands, relation names and attribute names. Then, we multiply each token with its corresponding weight index and eventually sum up all multiplications to yield the estimated misuse cost value for the query itself. Basically, we use Equation 3 for the calculation of M_Q, where SQL denotes the SQL command, w_{R_i} denotes the risk value assigned to the i[th] relation R_i, and $w_{A_{i,j}}$ denotes the risk value assigned to the j[th] attribute of the i[th] relation.

$$M_Q = w_1 \times SQL + w_2 \times \sum_{i=1}^{n} w_{R_i} + w_3 \times \sum_{i=1}^{n} \sum_{j=1}^{m} w_{A_{i,j}} \tag{3}$$

The queries in our system are in the form of SQL queries, consisting of basic or compound SQL commands. To make sure that we completely include the whole set of SQL commands, we used a comprehensive list of them and assigned a predetermined weight index to each. While assigning the weight indices, we take the amount and type of information a command is querying into consideration. For example, the SQL command SELECT is assigned a less weight index than that of the UPDATE or ADD commands in our design, because SELECT only reads data but ADD and UPDATE commands modify them. We repeat the risk weight assignment process for each relation and attribute in the database. To obtain a reliable weight assignment, a thorough analysis of the whole database is needed. For example, in our sample international organization database, querying a table containing employee business trips is less risky than querying a table storing information about employee performances. For this reason, we assign a lower risk weight to the business trips table than that of the employee performance table.

Likewise, for each attribute in the database, we determine how risky it would be to reveal (or modify) that field and accordingly assign a weight index to it. Once the weight assignment is complete, we can reuse it for future evaluations.

Syntactically, the SQL queries are made up of multiple clauses. A SELECT query, for example, has three clauses:

```
SELECT attribute name(s)
FROM relation name(s)
WHERE condition(s)
```

So, for every SELECT command, we calculate the risk value of each clause by multiplying each weight index. Then, we sum the risk value of each clause to calculate the risk for the whole SQL command (Equation 3). We repeat the same procedure for calculating the risk values for other SQL commands.

3.2.1 Query Representations

One of the inputs to our system is represented by the queries submitted by users. The queries can either belong to the current user, or to earlier users. Independently of its owner, a query is in the form of SQL statements. This feature helps us represent database queries in a standard, and more importantly in a shorter manner. We employ a query representation scheme having three different levels of granularity, which is similar to that of Bertino et al.'s in [4]. In the following three subsections we define these query representation schemes in detail.

3.2.2.1 Coarse Grain. This representation has the simplest level of granularity. Given a standard SQL query, it transforms it to the new format as: This representation has the simplest level of granularity. Given a standard SQL query, it transforms it to the new format as:
$\langle SQL\,Command, Relation\,Counter, Attribute\,Counter \rangle$, where a given SQL command is symbolized with a three-component scheme: (1) the name of the SQL command, (2) the number of relations involved in the command and (3) the count of attributes involved in the command.

When this scheme is incorporated to represent input queries, we use Euclidean distance or Hamming distance to calculate distances from cluster centroids. Euclidean distance is a general metric for measuring the distance between two points in a multi-planar space. For points A=(a_1, a_2, .., a_n) and B=(b_1, b_2, …, b_n), the Euclidean distance between them is calculated as:

$$d_{Euclidean} = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2} \tag{4}$$

We use Euclidean distance for coarse grain and medium grain query represent-tations. Since the fine grain representation of queries is simply a binary represent-tation, we use Hamming distance to calculate the distances from cluster centroids for such query representations. In essence, Hamming distance is a special form of the Euclidean distance and allows faster calculation.

3.2.1.2 Medium Grain. This representation has finer granularity as compared to the coarse grain technique. It represents SQL queries in the following format:
$\langle SQL\,Command, Attribute\,Counter[\,] \rangle$ where the first component is the name of the SQL command and the second component AttributeCounter[i] contains the number of attributes of the ith relation in the SQL command. This is a modified representation of Bertino et al.'s corresponding (m-triplet) format [4]. In their work, the authors symbolize the SQL command as a triplet by adding a binary bit vector of size equal to the count of relations in the database. We simplify this notation by removing the bit vector. Because, the attribute counter itself already signifies how many relations exist in the database.

As the case with coarse grain, we use the Euclidean distance with medium granularity to calculate distances from cluster centroids.

3.2.1.3 Fine Grain. As the name implies, this is the finest level of granularity in our implementation. This granularity represents a given SQL command with the new format as:

$$\langle SQL\ Command, Attribute\ Matrix[\ ][\ ] \rangle$$

where the second component is a binary matrix with the following rule:

$$AttributeM\ atrix[i][j] = \begin{cases} 1\ if\ \ jth\ attribute\ \ of\ \ the\ ith\ relation\ \ is\ accessed \\ 0\ otherwise \end{cases}$$

This representation has a modification to the f-triplet mode of Bertino et al.'s [4]. We remove the binary bit vector of size equal to the count of relations in the database due to the same reason given in subsection 3.2.1.2.

Since fine grain representation contains the AtrributeMatrix in a binary format, we use the Hamming distance metric to calculate distances from cluster centroids. This method is easier to implement and yields better performance in terms of speed on a binary matrix.

Throughout our implementation, we employ each of the above mentioned granularities to represent the set of current user queries as well as role history queries on real world database. As part of the application, we measure how successful each granularity is in determining the query clusters. Besides, we calculate the distance of each individual query belonging to the current user to each cluster. We then compare results to determine which granularity scheme best resembles the real world. Section 4 gives the experimental results on the real data set.

3.3 Role History Log (H)

In Equation 2, the third risk factor that is used to compute the probability of role misuse is the role history log (*H*). It consists of individual user queries that were submitted before. An important property of the role history contents is that, they are the queries which were granted access before. The only possible way for a query to be included into the history log is to obey the predefined role definitions. So, the history contains queries with considerably lower level of overall risk, what we call as normal queries.

We make use of data mining to generate clusters out of role history logs (*H*), so that we categorize what type of behaviors users had before for the same role. Then, we refer to the current user's submissions and obtain individual queries (*Q*) from the set of current user queries (S_{CQ}) to calculate their distances from the role history clusters. In this manner, we determine how different the current user behaves from previous users. At the end of assigning a risk value for the current query, we add it to the role history log. This is another aspect that makes our design dynamic and adaptive.

For our ongoing international organization example, imagine that countries A, B and C had established another coalition –say coalition CBA- before. And assume that companies from countries A, B an C have had such queries recorded before into the history logs as: "What will be the name of our product?", "How many people will be working in our factory?", "How much money does country A/B put in the joint

project?", etc. We see that the question "How much money does country A/B put in the joint project?" in role history is exactly the same as the one asked in by the current user (Section 3.2.). So, by analyzing each individual query in set S_{CQ}, we determine if they are in "acceptable" distance from history clusters or not. Afterwards, we take an average of the distances to assign a single value indicating the risk contribution for the whole set S_{CQ} for the current user.

3.3.1 Using Data Mining

Data mining is applied to accomplish two fundamental types of tasks: The former type is called predictive and is used to estimate the value of a particular attribute based on other attributes. The latter type is called descriptive and it aims at deriving patterns so as to predict future behaviors [19, 24]. In our work, we employ descriptive data mining. More specifically, we utilize the K-means clustering for anomaly detection. It helps us form clusters to categorize historical data and then to determine how far (or close) the recent data to each cluster. In K-means clustering, the mean of a group of points determines a cluster centroid. By calculating centroids for each group, we get the cluster centers.

Choosing the parameter k is the key point for the success of K-means clustering algorithm. To determine the best possible k, we use the v-fold cross validation technique [14, 22]. Since the value of k is not known a priori, we first divide the overall data set into v different segments (folds). We use *v-1* segments as the training set and the v^{th} segment as the application set. We next apply the analysis to the *v-1* segments and then apply the results to the v^{th} segment. By repeating this procedure v times for each fold, we calculate an overall average and set the value of clustering parameter k accordingly. While doing this, we use the distance of each cluster from each other as the decision criteria. In our work, we set v=10 and l=9 for determining k=5.

In our work, we utilize Weka knowledge analysis tool for implementing the K-means clustering algorithm [23]. We derive query clusters from role history logs. By applying v-fold cross validation, we determine the optimum k value for the number of clusters, instead of Weka's default value of k=2.

K-means clustering distributes points into clusters in a two-dimensional space. Several distance metrics exist in clustering to calculate distances from cluster centroids: Euclidean, squared Euclidean, city-block (Manhattan), Chebychev, power distance and performance disagreement. Among them, we choose the Euclidean distance because it considers both dimensions and neutralizes their effect by first squaring the distance, then by taking the square root of it.

3.4 Expected Utility (U)

An important risk factor that contributes to the calculation of the total expected return (Equation 1) in our system is the expected utility (U). In the international organization example, the expected utility is the profit that is expected by setting up the factory in country C. Let's assume that the city in country C where the new factory is to be founded is one of the most unsafe cities in this country. So, setting up the new business there incurs a high risk regarding factory workers' security. On the other hand, this city has very fertile soil to grow the major material that is needed for the

product –say product X- that will be produced in that factory. Then, the coalition ABC may still take the risk and set up the factory in country C just because the expected utility is very high.

During implementation, we assume that the expected utility for the current query is given as a fixed value.

4 Experimental Results

We run our program on a real world data set [4]. In our current implementation, we incorporate our risk model to calculate the level of risk in "role to permission" assignments in RBAC. In our risk model, all the factors except the misuse probability (P) are given as parameters to our system. Therefore, in the following subsections, we focus only on measuring P in our experimentations.

4.1 Data Set Definition

To implement our design, we used a real data set containing SQL queries submitted in a database of a medical clinic [4]. The database consists of 130 relations with totally 1201 attributes. The query log has 7588 instances that were submitted by users belonging to one of 8 different types of roles in the database.

4.2 Implementation

We implemented three different granularities as coarse, medium and fine on the data set. In order to determine the value of k for K-means clustering, we applied 10-fold cross-validation. This work yielded the best results for k=5. Setting k=5 in K-means algorithm, we first determined the cluster centroids for each role. Then, we ran our scheme on individual user queries in each role in a sliding window basis to calculate the distance of each current user query from each cluster centroid. We anticipate that set of current user queries are close to the clusters belonging to his actual role definitions, but with different role definitions, we expect our design to yield longer distances to the cluster centroids.

While determining the level of risk, we make use of the probability that a user belonging to Role X behaves as if he is a user belonging to Role Y. We call this the role misuse probability (P), whose formula is given in Equation 1. In an ideal system, such role intersections are expected to be very few for security purposes. This requires the role misuse probability to be as low as possible. If this probability is high, then we assign a high risk value for the query. The calculation of the misuse probability involves the normal probability distribution of the minimum distance to the cluster centroids that we obtained earlier with K-means clustering. We set $P = 1 - \phi(d_Q)$, where is the misuse probability, ϕ is the probability density function (PDF) of normal distribution, and d_Q is the distance value for the query Q to its nearest cluster centroid. The normal distribution PDF computes $\phi(d_Q)$ by using the mean (μ) and standard deviation (σ) values. We repeat the risk calculation steps for each query belonging to a single user to compute the overall risk for him. Once

parameters are set, the risk evaluation becomes a repetitive routine and can be accomplished in a straightforward and easy way.

The database we used in our experiments is considerably large with 130 relations and 1201 attributes. So, employing coarse grain representation involves two columns only, while medium grain involves as many columns as the number of relations (130) and fine grain representation involves as many columns as the number of attributes (1201) for the current application. Handling so many columns for misuse probability calculation is computationally very expensive. The runtime for medium grain and fine grain approaches were in the order of several ten minutes. For this reason, we used coarse grain representation to calculate distances and the misuse probabilities in the ongoing experiments.

We conducted two sets of experiments for the calculation and interpretation of misuse probabilities while setting Role 0 as the base role for both cases. In the former, we calculated the distances of all queries in roles other than Role 0 to the base role. In the latter, we conducted the same experiments to calculate the distances of queries in Role 0 to the base role, i.e. Role 0. The role group that contains the largest number of queries is Role 0 for the current data set. For this reason, we set Role 0 as the base role in our experiments. The reason why we combined the queries belonging to all role groups except for Role 0 is that, most of the individual role groups have very few (even single) queries. So, the base role Role 0 has 6170 queries and the rest of the role groups 1, 2, 3, 4, 5, 6, and 7 have 4+20+104+1+156+10+1123=1418 queries in total.

In the first part of the implementation, we first formed the mixture group of queries from role groups 1, 2, 3, 4, 5, 6 and 7. To obtain misuse probabilities, we first implemented K-means clustering to find the cluster centroids in the base role, and then we used the distance of each query (d_Q) to calculate the average distance to the nearest centroid in Role 0 (μ) and its standard deviation (σ) for this role. Assuming that the queries from Role 0 show a normal pattern, we used the population mean (μ) and standard deviation (σ) for the calculation of the normal distribution probability, where the population is the whole set of query distances in Role 0. Table 1 lists the results we obtained.

Table 1. Misuse probabilities of all queries with base role=Role 0

a.) from Roles 1, 2, 3, 4, 5, 6, 7

Probability Group	Distance (d_Q)	Probability (1-Φ(d_Q))
AllRolesExcept0_g1	0.23	0.8809
AllRolesExcept0_g2	0.26	0.8801
AllRolesExcept0_g3	0.74	0.8690
AllRolesExcept0_g4	0.77	0.8685
AllRolesExcept0_g5	1.01	0.8643
AllRolesExcept0_g6	1.26	0.8607
AllRolesExcept0_g7	1.74	0.8570
AllRolesExcept0_g8	1.90	0.8567
AllRolesExcept0_g9	2.10	0.8570
AllRolesExcept0_g10	2.33	0.8583
AllRolesExcept0_g11	2.74	0.8628
AllRolesExcept0_g12	2.90	0.8652
AllRolesExcept0_g13	2.92	0.8656
AllRolesExcept0_g14	4.02	0.8921
AllRolesExcept0_g15	6.03	0.9518
AllRolesExcept0_g16	6.40	0.9606
AllRolesExcept0_g17	8.38	0.9903
AllRolesExcept0_g18	9.68	0.9970
AllRolesExcept0_g19	10.10	0.9981

b.) from Role 0

Probability Group	Distance (d_Q)	Probability (1-Φ(d_Q))
Role0_g1	0.23	0.8959
Role0_g2	0.26	0.8953
Role0_g3	0.74	0.8860
Role0_g4	0.77	0.8855
Role0_g5	1.26	0.8784
Role0_g6	1.34	0.8774
Role0_g7	1.59	0.8751
Role0_g8	1.74	0.8740
Role0_g9	1.90	0.8732
Role0_g10	2.01	0.8728
Role0_g11	2.33	0.8726
Role0_g12	2.74	0.8742
Role0_g13	2.90	0.8754
Role0_g14	2.92	0.8756
Role0_g15	4.02	0.8919
Role0_g16	6.03	0.9392
Role0_g17	6.40	0.9476
Role0_g18	8.38	0.9816
Role0_g19	9.68	0.9925
Role0_g20	10.10	0.9947
Role0_g21	17.01	~1.0000
Role0_g22	20.00	~1.0000

In Table 1a, we list the misuse probability values for different distance groups together with each group's distance value. According to the table, there are 19 such distance groups based on the values. When we look at the probability values, we see that as the distance gets higher, the probability value increases, as was expected.

We repeated the same experiment to measure the misuse probabilities of queries in Role 0 to itself, i.e. to Role 0, and we list them in Table 1b.

Our expectation was that, the misuse probabilities for Role 0 to Role 0 (Table 1b) would be much lower than that of all other roles to Role 0 (Table 1a). Results show that the misuse probabilities for Role 0 to Role 0 are still very high, being very close the values listed in Table 1a. The reason for that is the occurrence of the same or similar queries in Role 0, as well as in other roles in the data set. For example, the SQL query in Role 0 as:

```
SELECT  check_in_date, planed_start_time, contract_no, treatment_id,
treatment_consultant, branch_id
FROM treatment_schedule
WHERE customer_id = '100300199' and treatment_status = 1
ORDER BY check_in_date desc;
```

occurs very similarly in Role 1 as:

```
SELECT  check_in_date, planed_start_time, contract_no, treatment_id,
treatment_consultant, branch_id
FROM treatment_schedule
WHERE customer_id = '100200072' and treatment_status = 1
ORDER BY check_in_date desc;
```

and in Role 2 as:

```
SELECT  check_in_date, planed_start_time, contract_no, treatment_id,
treatment_consultant, branch_id
FROM treatment_schedule
WHERE customer_id = '100201056' and treatment_status = 1
ORDER BY check_in_date desc;
```

and in Role 7 as:

```
SELECT  check_in_date, planed_start_time, contract_no, treatment_id,
treatment_consultant, branch_id
FROM treatment_schedule
WHERE customer_id = '100300499' and treatment_status = 1
ORDER BY check_in_date desc;
```

The only difference in the four queries listed above is the customer_id number that is queried, whereas for the rest the queries are the same. So, the representation of each query in different role groups would be exactly the same in coarse, medium and fine grain, respectively. Likewise, the SQL query in Role 0 as:

```
SELECT  *
FROM contract_record
WHERE contract_no = 'm2810'
```

and another query in Role 0 as:

```
SELECT  contract_date, contract_no, outstanding_balance, active_status
FROM contract_record
WHERE customer_id= '100201496' and  contract_type =0 and (active_status
= 0 or outstanding_balance <> 0) and active_status <> 2
ORDER BY contract_date desc;
```

occurs in Role 7 as:

```
SELECT  *
FROM contract_record
WHERE contract_no = 't4596';
```

and again in Role 7 as:

```
SELECT  contract_date, contract_no, outstanding_balance, active_status
FROM contract_record
WHERE customer_id= '100300951' and  contract_type =0 and (active_status
= 0 or outstanding_balance <> 0) and active_status <> 2
ORDER BY contract_date desc;
```

multiple times. There are several such occurrences that would eventually lead to the same query representation for different roles in the data set. Consequently, a more precise representation for the queries may be needed for a better performance of our risk model.

According to the risk model we introduce in order to lower the level of risk, the total expected return (E) of Equation 1 should be greater than 0. Thus, the inequality $\frac{(1-P)}{P} > \frac{M_Q}{U}$ must hold. Our experimental results in Tables 1a and 1b indicate that if the role definitions have significant overlap, we end up with high misuse probabilities for all roles. Based on the calculated misuse probabilities, the inequality above implies that the expected utility (U) must be at least 7 times larger than the misuse cost (M_Q) on the average to make the total expected return (E) positive.

As the experimental work shows the role boundaries are not distinct in the real world data set we use, which leads to role overlapping. Apparently, the distribution of queries among roles in this data set is not well designed. Additionally, query distribution among roles is significantly uneven. With an automatically generated synthetic data set, one can most probably obtain more reasonable results with the risk evaluation model we propose.

5 Conclusions and Future Work

In this paper we propose a quantitative model to measure risk of role misuse in RBAC employed distributed environments. Our design is an extension to the well known standard access control model called RBAC. Even if RBAC provides a comprehensive infrastructure, it does not consider the amount of risk involved in granting access requests. This risk is incurred by role misuse. We design and implement a risk model to complement RBAC for enhanced access control. Our risk calculation scheme is based on a statistical utility function. For that, it uses the risk factors as user credentials, set of current user queries, role history logs and expected utility. Our architecture is flexible enough to be placed in user to role or role to permission assignments in the RBAC model.

To represent queries we incorporated three different granularities and compared their performances. Due to the large amount of relations and attributes in the data set, the coarse grain approach yielded the best results. We also utilized data mining to find out different user clusters based on role history. We then determined how far each individual query of the current user is from these clusters.

Implementation of our scheme on a real data set showed that there are two main sources of risk for a distributed database environment: (1) the inherent risk incurred by user credentials, (2) the risk caused by role misuse. It is considerably easier to manage the former case because role assignments are possible only after ensuring that

the user credentials comply with the database requirements. In the latter, we can detect role misuse only when role assignments are already made, which is usually too late.

Since the K-means clustering results did not yield the expected results in terms of query classification among roles, we suggest that the risk management scheme should focus more on the determination of the nature of query that we proposed in Section 3.2. With regards to the experimental work, we see that the most part of the risk is sourced from the nature of the query, i.e. how many attributes a query attempts to access and what are these attributes. So, to measure the risk properly we need to tokenize and analyze user queries individually.

As part of a broader solution, we also suggest minimizing role definitions in the database, if database intervention is possible. In that case, while we keep the number of tasks constant, we increase the number of roles in the system to accomplish these tasks. This, in turn, means assigning multiple roles to individual users. Also, we propose employing query templates in the database to prevent overlapping roles.
Considering the two main sources of risk, we need to take precautions for each source. For credentials, the risk evaluation should consider when, where, and how frequently a user is connecting for sending queries. So, we need to bring strict controls over credentials. For the role misuse, we should analyze the attributes that each role can access.

To obtain better results, we will search data sets that will fulfill the requirements of our design. Such a data set needs to have well defined and differentiated roles and distinct queries that are evenly distributed among roles. If we cannot find a real data set as requested, then we will generate synthetic data to test our system or reorganize the existing data set. Furthermore, we will employ K-means clustering for other values of k (e.g. k+1, k+2, etc.) than 5. We will also search for other data mining techniques (e.g. EM algorithm) to cluster the query histories. By comparing the new results with our current implementation, we will determine the optimum data mining algorithm. We will also investigate alternative ways of representing the database queries. We plan on expanding our work to determine what to do after risk evaluation. Our preliminary suggestion is to employ role encryption if the risk is too high. By encrypting the role information, we expect to strengthen the accountability of the system, hence to ensure better security.

Acknowledgements

The work reported in this paper is part of the project "Systematic Control and Management of Data Integrity, Quality and Provenance for Command and Control Applications" partially funded by the USA Air Force Office of Sponsored Research.

References

1. Abdul-Rahman, A., Hailes, S.: Supporting trust in virtual commmunities, Hawaii International Conference on System Sciences, Hawai, USA (January 2000)
2. Anderson, J.F., Brown, R.L.: Risk and Insurance, Number 1-21-00 in Study Notes, Society of Actuaries (2000)

3. Bacon, J., Dimmock, N., Ingram, D., Moody, K., Shand, B., Twigg, A.: SECURE Deliverable 3.1: Definition of Risk Model (December 2002)
4. Bertino, E., Kamra, A., Terzi, E., Vakali, A.: Intrusion detection in RBAC-administered databases. In: 21st Annual Comp. Sec. Applc Conf., Tucson, AR, USA (December 2005)
5. Cahill, V., Wagealla, W., Nixon, P., Terzis, S., Lowe, H., McGettrick, A.: Using trust for secure collaboration in uncertain environments. IEEE Pervasive Comp. 2, 52–61 (2003)
6. Carbone, M., Dimmock, N., Krukow, K., Nielsen, M.: Revised Computational Trust Model, EU IST-FET Project Deliverable (2004)
7. Dimmock, N.: How much is enough? Risk in trust-based access control. In: IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises - Enterprise Security, pp. 281–282 (June 2003)
8. Dimmock, N., Belokosztolszki, A., Eyers, D., Bacon, J., Moody, K.: Using trust and risk in role-based access control policies. In: 9th ACM Symposium on Access Control Models and Technologies, Yorktown Heights, New York, USA (June 2-4, 2004)
9. Dimmock, N., Bacon, J., Ingram, D., Moody, K.: Risk models for trust-based access control (TBAC). In: Herrmann, P., Issarny, V., Shiu, S.C.K. (eds.) iTrust 2005. LNCS, vol. 3477, Springer, Heidelberg (2005)
10. English, C., Wagealla, W., Nixon, P., Terzis, S., Lowe, H., McGettrick, A.: Trusting collaboration in global computing systems. In: Nixon, P., Terzis, S. (eds.) iTrust 2003. LNCS, vol. 2692, pp. 28–30. Springer, Heidelberg (2003)
11. Ferraiolo, D., Kuhn, R.: Role-based access control. In: 15th NIST-NSCS National Computer Security Conference, pp. 554–563 (1992)
12. Gallaher, M.P., O'Connor, A.C., Kropp, B.: The Economic Impact of Role-Based Access Control, Planning Report 02-1 for NIST, NC, USA (March 2002)
13. Joint Staff, Net-Centric Operational Environment Joint Integrating Concept, Washington, DC, USA (October 2005)
14. McLachlan, G., Peel, D. (eds.): Finite Mixture Models. Wiley and Sons, USA (2000)
15. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role based access control models. IEEE Computer 29(2), 38–47 (1996)
16. Schlegelmilch, J., Steffens, U.: Role mining with ORCA. In: Proceedings of the 10th ACM Symp on Access Cont. Models &Techn., Scandic Hasselbacken, Stockholm (June 1-3, 2005)
17. Shand, B., Dimmock, N., Bacon, J.: Trust for ubiquitous, transparent collaboration. Wireless Networks 10, 711–721 (2004)
18. Smith, T.: Information risk: a new approach to information technology security, IT Solutions [Accessed July 18, 2006], http://itsolutions.sys-con.com
19. Tan, P.N., Steinbach, M., Kumar, V.: Intro. to Data Mining, Pearson Education, USA (2006)
20. Thuraisingham, B.: Information Operations Across Infospheres, Annual Report for Air Force Office of Scientific Research (October 2006)
21. Vaidya, J., Atluri, V., Warner, J.: RoleMiner: mining roles using subset enumeration. In: 13th ACM Conf. on Computer & Comms. Security, Alexandria, VA, USA (October 2006)
22. V-fold Cross-Validation [Acc. October 26, 2006], http://www.statsoft.com/textbook/stcluan.html
23. Weka [Accessed October 26, 2006], http://www.cs.waikato.ac.nz/ml/weka
24. Witten, I.H., Frank, E.: Data Mining. Morgan Kauffman Pub, USA (2000)
25. Xiong, L., Liu, L.: Building trust in decentralized peer-to-peer electronic communities. In: Fifth International Conference on Electronic Commerce, Pittsburgh, PA, USA (October 2003)
26. Zhang, C.N., Yang, C.: An object-oriented RBAC model for distributed system. In: Working IEEE/IFIP Conference on S/w Arch. (August 28-31, 2001)