Risk and Decision Analysis 1 (2009) 21–33
DOI 10.3233/RDA-2008-0002
IOS Press

A risk management approach to RBAC

Ebru Celikel [a], Murat Kantarcioglu [a], Bhavani Thuraisingham [a] and Elisa Bertino [b]
[a] *University of Texas at Dallas Richardson, TX 75083, USA*
[b] *Purdue University, West Lafayette, IN 47907, USA*

Abstract. Even if Role Based Access Control (RBAC) is employed properly, distributed database environments are still prone to illegitimate access attempts: in RBAC, users potentially carry the risk of illegal access attempts via credentials violation, or unintentional/intentional incorrect use of already granted permissions via role misuse/abuse. We introduce a probabilistic risk management model for enhanced access control in such databases. We incorporate failure modes and effects analysis (FMEA) scheme for measuring user risks in our design. We combine components as credentials, queries, role history logs and expected utility for a probabilistic formulation of risk. We present experimental results that we obtained on real world database. The results emphasize the need for a database where roles are well defined and queries under different roles do not overlap. We suggest using query templates with minimized role definitions so that the risk model we introduce provides better risk management.

Keywords: Risk management, RBAC, access control

1. Introduction

The risk of an event is the potential to affect the outcome in a positive or negative manner [1]. By definition, risk is closely related with uncertainty. Hence, a model to measure risk needs to handle the uncertainty involved. In this study, we propose a risk management system to manipulate risks in RBAC employed distributed environments. The risk management scheme we introduce consists of two main components as risk assessment (analysis) and risk control, where the former is related to the identification of vulnerabilities, as well as probability and consequence assessment; and the latter refers to the definition of acceptable level of risk and consequence mitigation in the system.

Database environments have users, data and technological means as the key components. A good database management system is supposed to provide a comprehensive administration of these components. Among these components, data is the most vulnerable: it is threatened either by illegal access, or by legal but incorrect use, which constitute the two main sources of risk in the database. For the most part, these risks are brought to the system by users. This is because user behavior is mostly unpredictable.

Role Based Access Control (RBAC) has been introduced in an effort to prevent these risks from occurring, and/or to reduce their negative effects when they occur. By restricting user permissions to predefined role definitions, RBAC provides a satisfactory level of security on database administration. Especially for distributed database environments, where users can access database from various physical locations, access granting and database security becomes very important. If users attempt to access the database illegally, then this is a violation of database security and RBAC handles this by requesting credentials from each user. If they do not conform to the system requirements, then RBAC will immediately refuse the database connection attempt. Even if user credentials are valid and access right is granted legitimately, users may still violate the database security: they submit illegitimate queries either intentionally or unintentionally, which leads to the case that we refer to as role misuse. At this point, RBAC may remain inadequate and the database system is under risk.

In this work, we design and develop a risk management model for RBAC employed distributed database environments. We expect our design to analyze and control risk in such databases.

2. Related work

Although risk is a general term, assessment and manipulation of risk in databases and more specifically in RBAC is considerably a new topic. One of the few recent works on the consideration and manipulation of risk belongs to Dimmock et al. The authors develop a framework called SECURE (The Secure Environments for Collaboration among Ubiquitous Roaming Entities) [2,5,7,6,8–11,18] for establishing a trust-based generic decision making. As a basement for this framework, they use an extended RBAC implementation called OASIS (The Open Architecture for Secure Interworking Services) and apply SECURE on a grid computing environment. This study introduces a general trust model by focusing on the other principal's trustworthiness – denoted as likelihood – and the outcome's cost. In this format, the SECURE framework is a general trust scheme rather than a risk model with two parameters as likelihood and outcome. Their work is extended by Carbone et al. [7,6] to enclose a formal trust model that concentrates on the aspects of trust formation, evolution and propagation in a global computing environment. With global computing they refer to a platform where entities are autonomous, decentralized, mobile, dynamically configurable, and flexible enough to operate on partial information. Different from the SECURE project and its successives, our work focuses on risk by introducing a third parameter to its calculation: detection rating. Furthermore, we base our design specifically on RBAC employed distributed databases rather than proposing a more general risk model. Consequently, the major component we use for risk calculation is the user queries.

In another work, Nissanke and Khayat [15] introduce a security risk ordering on tasks involving different access operations posed by different users in RBAC. With a graph based representation of risk ordering, they try to solve the permission delegation and allocation problem for users by making use of role hierarchy principles. Their work singles out the actual risk assessment itself. They assume that the outcome of the risk evaluation process is given. The authors then build a system to combine the following: the formulation of several principles to define role hierarchies and to manipulate role delegations. Our proposal can be considered as a front-end to this work: providing a complete risk model including risk calculation and control.

3. Risk management scheme

Our aim is to design and develop a risk management model to measure and evaluate user risks in a distributed database where RBAC is employed. The risk management system we propose is made up of two key components as risk analysis (assessment) and risk control. During risk analysis we identify risks, and assess event probabilities together with consequences. With risk control, we define the tolerable level of risk, compare and evaluate alternatives via monitoring and decision making and provide failure prevention and risk mitigation. With risk mitigation, we refer to the actions to reduce the likelihood of a negative event occurring or to reduce the negative consequences if the event occurs.

Risk analysis in a database is important because it helps quantify and model risks caused by various factors under several conditions. The analysis of risk also assists in decision making and determination of the cost-benefit tradeoff process in the database.

In general, the systematic approach for risk analysis involves the following: the definition of objectives and failure scenarios, collection of data, assessment of qualitative and quantitative risk, failure prevention and risk mitigation. Several risk scenarios may exist in a database: e.g. risks caused by technical problems, management risks, and user risks. In our design, we primarily concentrate on user risks. User risks are incurred to the system either by users having illegitimate credentials, or by users that are already authorized but use their access rights in an incorrect manner.

In Fig. 1, we give a flowchart of the risk management system that we design and develop.

In the following subsections, we define the individual steps of the risk management model we introduce.

3.1. *Definition of database system*

As given in Fig. 1, we define the database system under consideration in the very first step of our

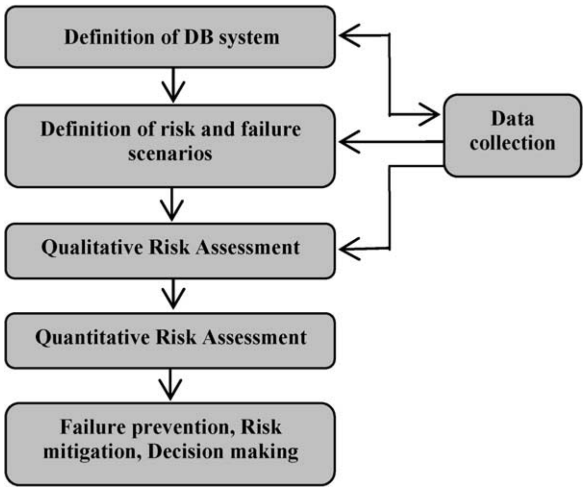
Definition of DB system
Definition of risk and failure scenarios
Qualitative Risk Assessment
Quantitative Risk Assessment
Failure prevention, Risk mitigation, Decision making
Data collection

Fig. 1. Risk management scheme flowchart.

risk management design. Database definition combines several processes as identifying risk analysis and control objectives, determining boundaries, and identifying success criteria in terms of measurable performances. During the course of database system definition, we employ data collection. The data collection contains information to evaluate failure likelihood and consequences.

3.2. *Definition of risk and failure scenarios*

At the second step of our risk management system design, we define risk scenarios (Fig. 1). For a general system, several sources of risk – and accordingly many risk scenarios – are likely to occur. In this study, we concentrate on an RBAC employed distributed database environment rather than a general system. Even if we concentrate on a specific system, still the general case of risk occurrences holds: several sources of risk accompanied by multiple scenarios are probable for such a database. The generic model for RBAC is associated with three essential components: (1) user, (2) role and (3) permission. Among them, we deliberately focus on risks incurred by users. In a usual RBAC setting, users are assigned to roles which are then granted permissions to perform predefined tasks [12,13,16,22]. The reason why we focus on user component is that users are very likely to incur risks because they are the component of the database system that make several access attempts in various forms. Furthermore, defining and handling user risks would allow an early detection and control of probable negative consequences in the system.

Before we give a sample scenario, we make the following assumptions: we assume that the physical storage media where the database is stored is safe. More specifically, we assume that the database cannot be accessed in any other way than RBAC, i.e. no by-pass is possible.

For better understanding of the scheme, let's assume an RBAC administered distributed database where users are mostly located at geographically distant locations. Due to its distributed feature, such a database becomes more prone to security flaws as the number of users, roles and permissions increase. Therefore, we need to define probable failure scenarios to take necessary precautions on time. A failure scenario is the complement of a success scenario. Then, we start with defining a success scenario for the sample database: Assume that an arbitrary user say user A logs in to a dynamically maintained events calendar database under the role of public relations staff. User A wants to enter an announcement about an upcoming workshop event. He also wants to change the show time of a previously entered event and plans to list all events that will occur in July. To obtain access right to the database, the first thing user A needs to do is to submit his credentials to the events calendar database. At that point, the admin of the database investigates the institution that issued user A's credentials and checks his credentials for correctness. If the credentials are legitimate, user A is assigned to the role of public relations staff in the database. According to the predefined role boundaries, a public relations staff role in the database is allowed to make new entries about upcoming event announcements, to update the existing ones and to list the upcoming events. After user A's credentials are confirmed for correctness, he is allowed to submit his three queries, i.e. entering a new event announcement, changing the show time of an existing event and listing all upcoming events in July. As long as each individual query of user A complies with the public relations staff role definition in the underlying RBAC model, his queries are executed and results are returned. Even if a query complies with the role definition under which it is supposed to be executed, submitting it too many times should raise a risk condition warning, i.e. a role misuse. Also, a query attempt that tries to exceed its role boundary should raise another risk condition warning, what we call as role misuse. Subsequently, a success scenario for an ideal user would require legitimate user credentials together with no role misuse or role abuse attempt. In Fig. 2, we give a block diagram of the success scenario that we define.

To identify the risk scenarios we use the success tree approach, which represents the successful system operation as a logic expression. This is because success trees are a general means of representing risks in any system and they can visualize the effect of individual components even in complex environments.

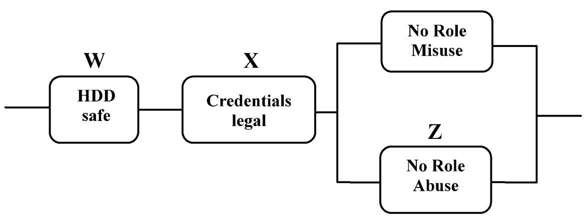

Fig. 2. Block diagram for the sample scenario.

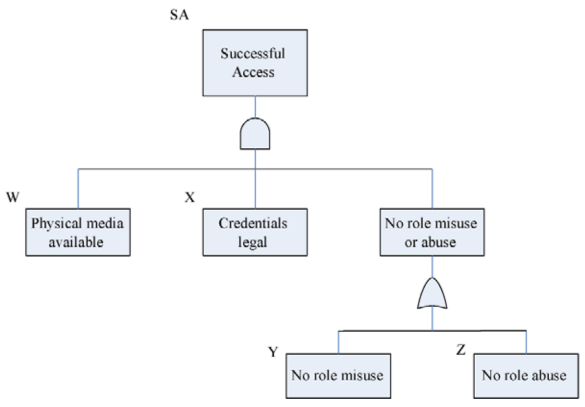

Fig. 3. Success tree of the sample scenario.

The success tree for our on going scenario is drawn in Fig. 3.

According to the success tree (Fig. 3) based on the sample scenario block diagram in Fig. 2, a successful database access would occur when the logic expression below holds true:

$$\mathit{Successful\ Access} = W \cap X \cap (Y \cup Z). \tag{1}$$

Hence, with simple conversions, the probability of successful access $P(\mathit{SA})$ becomes:

$$P(\mathit{SA}) = P(W) \times P(X) \times \bigl(1 - P(Y') \times P(Z')\bigr). \tag{2}$$

Our aim is to maximize the probability $P(\mathit{SA})$ so that the legitimate access attempts result in success and do not violate the security of RBAC.

3.3. *Qualitative and quantitative risk assessment*

According to Fig. 1, the definition of failure and risk scenarios step in our risk management scheme is followed by qualitative and quantitative risk assessments step. They both aim at determining the probability (P_i) and consequence (C_i) figures for each risk factor i. The qualitative risk assessment relies only on judgment and expert evaluations, a property that makes it insufficient when it is applied individually in our system. For this reason, we also incorporate quantitative risk assessment, which utilizes probabilistic and statistical methods for risk factor measurements in our implementation.

While determining which risk assessment technique to employ, we have considered several alternatives as preliminary hazard analysis (PrHA), failure modes and effects analysis (FMEA), fault-tree analysis (FTA) and event tree analysis (ETA). Among them, we have chosen FMEA. This is because it is a fast and practical method to assess risk and has been applied in several areas. Furthermore, the inductive property of FMEA well complies with our motivation: bringing multiple risk factors together to calculate the overall risk.

We consider two access control violations as potential sources of risk for the database: (1) illegitimate user credentials, (2) role abuse/misuse. In the former, the risk factor is the credentials of the user, and in the latter the query submitted by the user determines the risk. We need to tokenize a query to reveal what it really does: either adding a new instance to the database, or listing attribute(s), or updating attribute(s) or deleting attribute(s) and/or relation(s). Obviously, not every attribute in each relation, and every relation in the database would have the same degree of importance. We need to assign various degrees of importance to each factor by taking the consequence of its execution into consideration. This process requires a thorough analysis of the database.

Regardless of the system of consideration, the general risk formula involves multiple risk factors with different probabilities and consequences. Hence, the general risk formula can be represented as follows:

$$\mathit{Risk} = \sum_{i=1}^{n} P_i \times C_i, \tag{3}$$

where P_i denotes the probability of each risk factor $i = \{1, \ldots, n\}$, and C_i denotes the consequence (outcome) these risk factors.

For a distributed database environment with RBAC, several risk factors are probable. Among them, illegitimate credentials, role misuse and/or abuse attempts (that we call as user risks in short), malfunctioning database system and physical acquisition of the storage media are the most threatening. It is the responsibility of RBAC to ensure that only users with legitimate credentials are granted permissions to access requested database resources. So, the risk factor of illegitimate credentials is eliminated in advance by RBAC model. In our implementation, we assume that the physical protection of the storage media is handled by the risk mitigation efforts (which will be a part of the overall risk management system). So, illegal acquirement of the physical storage media problem is also eliminated and we do not take it further into consideration as a risk factor. Hence we mainly focus on the user risk and two factors in it that threaten the security of access in the database: (1) role misuse and (2) role abuse.

To process each risk factor, we utilize the FMEA method and suggest the use of risk priority number (*RPN*) that is defined as part of FMEA. *RPN* is a number related to a particular risk factor and is calculated as the multiplication of several ratings with weight assignments. We use *RPN* to determine the contribution of each risk factor to the computation of the overall risk. The general formula for the risk priority number (RPN_i) is as follows:

$$RPN_i = OR_i \times SR_i \times DR_i, \tag{4}$$

where OR_i denotes the occurrence rating, SR_i denotes the severity rating and DR_i denotes the detection rating for the risk factor i. RPN_i is an index that indicates how serious a risk factor is. So, the larger the RPN_i, the higher would be the severity of the risk factor. Each risk factor that contributes to the calculation of RPN_i is in the form of a rating scale, where a larger scale implies higher risk.

By definition, the above mentioned two formulas are closely related: for each risk factor i, the parameter probability (P_i) of Eq. (3) becomes the occurrence rating (OR_i) in Eq. (4), and the parameter consequence (C_i) of Eq. (3) becomes the severity rating (SR_i) in Eq. (4). Additionally, Eq. (4) introduces a new parameter to the risk formula: the detection rating (DR_i). The reason why we need this additional parameter is explained in the following paragraph.

Our aim is to compute a single risk value for each user. Hence, we assume $i = 1$ for Eq. (4). To establish a comprehensive risk evaluation scheme in our work, we concentrate on user risk that is caused by illegitimate credentials, role misuse and/or role abuse. Since RBAC eliminates the occurrence of illegitimate credentials, we are left with two main sources (factors) of user risk: role misuse and role abuse. To compute the user risk, we base our user risk calculation on queries and consider several aspects of a query into consideration: the frequency of a query, which is denoted by the occurrence rating (*OR*) parameter in Eq. (4), and the content of the query, which is denoted by the severity rating (*SR*) parameter of Eq. (4). According to the RBAC model, users and their queries can be under different role definitions. So, for a comprehensive risk calculation, we need to consider how similar a query is to the other queries in the same and/or other role definitions, and whether we can detect this easily or not. This brings the need for a third parameter for similarity detection, which is denoted by the detection rating (*DR*) of Eq. (4). Transforming Eq. (3) into Eq. (4) allows us to include more parameters into calculation of user risk.

As for the actual risk computation, for each individual user that submits quer(ies) to the system, we calculate a risk magnitude for him. In a typical database, users can submit multiple queries to the system and the risk of a particular user should be determined by considering each individual query he submits. For that reason, we use the set of queries (*SQ*) that are submitted in a predetermined time period (t) by a single user (U). For each query q from this set (*SQ*), we calculate its occurrence rating (*OR*), severity rating (*SR*) and detection rating (*DR*) and finally we multiply each to compute the risk priority number (*RPN*) for the current user. Below, we give details regarding the calculation of each parameter in *RPN* for each user.

3.3.1. Occurrence rating (OR)

Basically, we use the occurrence rating (*OR*) as a means of expressing the degree of role abuse in the database. A typical role abuse would occur when a user attempts to submit a query that he was already permitted to execute too many times. So, a simple method to detect the potential role abuse is to determine the number of occurrences of a particular query that is submitted by the same user. To calculate the occurrence rating (*OR*) for a single user, we refer to his query history logs so as to determine how many times this query has been submitted by the same user before. As was used in other implementations [1] we use a standard rate based on a scale of 10 for each RPN multiplicand: For *OR*, the highest rate is indicated with a 10 and the lowest rate is indicated with a 1. For our design, we use the *OR* listing given in Table 1.

According to Table 1, a high *OR* indicates a user with higher risk and low *OR* indicates a user with lower level of risk.

To compute the overall *OR* for the current user, we consider each distinct query in the whole set of queries (*SQ*) submitted by him and sum their individual fre-

Table 1

Sample occurrence ratings for the scenario

OR	Values	Explanation
Minor	1	Query occurrence frequency $< 1\%$
Low	2–3	$1\% \leqslant$ Query occurrence frequency $< 3\%$
Moderate	4–6	$3\% \leqslant$ Query occurrence frequency $< 15\%$
High	7–8	$15\% \leqslant$ Query occurrence frequency $< 35\%$
Extreme	9–10	$35\% \leqslant$ Query occurrence frequency $\leqslant 100\%$

quencies. For fast and practical manipulation of each query, we reduce different query types into three different types of representations: these types are coarse grain, medium grain and fine grain, in the order of increasing complexity. We adapted the aforementioned granularity types from Bertino et al.'s work [3] and modified each type to remove the inherent redundancy they originally contained. More specifically, for coarse grain we rephrase a query as:

$$\langle \mathit{SQL\ statement}, \#\mathit{of\ Relations}, \#\mathit{of\ Attributes}\rangle.$$

With medium grain the query representation becomes:

$$\langle \mathit{SQL\ statement},\ \mathit{Attribute\ Counter}[\,]\rangle,$$

where $\mathit{AttributeCounter}[i]$ contains the number of attributes of the ith relation in the SQL statement. And lastly with fine grain, we symbolize a query with the statement below:

$$\langle \mathit{SQL\ Statement},\ \mathit{Attribute\ Matrix}[\,][\,]\rangle,$$

where $\mathit{AttributeMatrix}[i][j]$ gets a value of 1 if the jth attribute of the ith relation is accessed by the *SQL Statement* and gets a value of 0 otherwise.

In Bertino et al.'s work, the authors design and implement an intrusion detection system based on the classification of user queries in an RBAC employed database. As for classification they incorporate the Naïve Bayes Classifier algorithm. In that sense, our approach has some similarities with this work: as was done by Bertino et al., we use SQL statements as the basis of our design and utilize query classification to generate query clusters for further analysis. Our work also differs from this work: Bertino et al. aim at detecting intrusions in the RBAC employed database, while we focus on detecting queries that carry higher risk than that of what we call the normal queries. Moreover, Bertino et al. employ Naïve Bayes Classifier [4], which is a supervised classification algorithm, whereas we use K-Means [19], being an unsupervised classification algorithm, for the generation of query clusters.

3.3.2. *Severity rating (SR)*

Severity rating (*SR*) is a measure designating the degree of risk for the content of a query. This is what we call the nature of the query. *SR* is the second multiplicand of the general *RPN* formula (Eq. (4)) and is calculated very similar to how the occurrence rating (*OR*) is calculated. For the set of queries (*SQ*) that belongs to the current user, we process each query individually to reveal *SR*. For that, we first tokenize each individual query to find out its three components (tokens) as (1) SQL statement, (2) attribute(s) list and (3) relation(s) list. Clearly, not every token would participate equally in *SR* calculation because some queries only list attributes while others add new data to the database or modify the existing ones. To reflect such query differences, we use separate weight index tables for each of the three tokens as query type, attribute and relation. To find out the corresponding weight index of a token, we employ simple table lookup. Obviously, more complex queries having multiple attributes and/or relations will require comparatively more table lookup. But again, it does not necessarily mean higher weight calculations. This is because what constitutes the query nature is its quality, not its quantity. Finally, we add each component's contribution to calculate the overall risk value. Afterwards, we use Table 2 to find the corresponding *SR* value for this risk. This way, the *SR* reflects the type of the query, how many attributes and tables it involves and how sensitive is the data that it is associated with.

Table 2
Sample severity ratings for the scenario

Rating	Values	Explanation
Minor	1	SQL command risk < 1.000
Low	2–3	$1.000 \leqslant$ SQL command risk < 2.000
Moderate	4–6	$2.000 \leqslant$ SQL command risk < 3.000
High	7–8	$3.000 \leqslant$ SQL command risk < 4.000
Extreme	9–10	$4.000 \leqslant$ SQL command risk < 5.000

As for the *SR*, we use a scale of 10 as we did with *OR*. Again a high *SR* indicates a user with higher risk and low *SR* indicates a user with lower level of risk (Table 2).

3.3.3. *Detection rating (DR)*

We use detection rating (*DR*) in our design to measure the degree of role misuse. We define role misuse as the case where a user under role A submits queries under the role definition B. In an ideal RBAC environment, we expect role definitions to be well defined so that no single role overlaps with other role(s) in the database. But in reality this almost never happens and this leads to role misuses. We compute a single *DR* for the whole set of queries (*SQ*) that belong to the current user. This is different than that of *OR* and *SR* calculations. We first apply K-means clustering on query history to generate query clusters for queries under other role definitions. Then we compute the distance of *SQ*

Table 3

Sample detection ratings for the scenario

Rating	Values	Explanation
Non-detection	10	$distMeasure < 1$
Very low	9	$1 \leqslant distMeasure < 10$
Low	7–8	$10 \leqslant distMeasure < 100$
Moderate	5–6	$100 \leqslant distMeasure < 1.000$
High	3–4	$1.000 \leqslant distMeasure < 10.000$
Very high	1–2	$10.000 \leqslant distMeasure < 100.000$

to each cluster. We call this distance d_Q. By using this distance value, we calculate the probability of belonging, which we denote as PB_Q, for each query to a particular cluster so as to determine if we can detect similar queries in the RBAC system. If d_Q is close to other role's query cluster centroids, it means that the current user is behaving as one of the users in other role definitions and we are having difficulty in detecting it. For being more precise, we use the measure

$$distMeasure = \frac{d_Q}{average\ dis\ \tan\ ces},$$

where average distances is the sum of all other users' distances to the cluster centroids. This is because we are interested in how the distance of the queries of a single user is as compared to all other queries of other users under that role. So, in our design a small *distMeasure* would lead to a poor detection capability, while a larger *distMeasure* would mean better role misuse detection. Hence, we assign a higher *DR* for poor detection capability and a lower *DR* for better detection capability (Table 3).

3.4. Failure prevention, risk mitigation, decision making

After determining the risk factors and their ratings, we need to decide on what actions to take against them, and even more importantly, what to in case they occur. Failure prevention deals with the first part, while risk mitigation comes with the second part. Decision making is associated with all these stages.

4. Implementation

We implement our risk model on a real world data set, which contains the query logs of a health center located in Toronto, Canada [3,21]. This database consists of eight different roles that build up the role set $R = \{0, 1, 2, \ldots, 7\}$, where distribution of queries under each role definition is listed as follows.

Table 4

Query distributions under each role

Role	# of queries	Proportion
0	6170	81.31
1	4	0.05
2	20	0.26
3	104	1.37
4	1	0.01
5	156	2.06
6	10	0.13
7	1123	14.80
Total	7588	100.00

As seen from Table 4, the distribution of 7588 queries under each role definition is uneven: while 81.31% of the whole set of queries submitted to the system are under role 0, only 0.01% of the total queries belong to role 4. This introduces a very high variance as 7.98 among query distributions for each role. Obviously, this feature has a negative effect on our implementation. Because, our risk measurement relies on the fact that role definitions are well formed with strict role boundaries and query distribution among roles is highly balanced.

4.1. Occurrence rating (OR) results

As part of our implementation, we calculate a risk value for an individual user by processing the set of queries ($SQ = \{q_0, q_1, q_2, \ldots, q_n\}$) that he submitted during the period of time t. The value of time t can be set to different values depending on the requirements. In order to determine the occurrence rating for query $q_i(OR_{q_i})$, we compare each individual query q_i from *SQ* with other queries in the database history log and reveal how many times it was submitted before. A high *OR* value, i.e. a more frequent query does not necessarily imply that the owner of this query carries a higher level of risk. Still, a high frequency will have a direct effect to the calculation of the user risk. This is because *OR* is one of the three multiplicands of the general formula to compute the overall risk, i.e. the risk priority number (*RPN*) in Eq. (4). Obviously, a high *RPN* means high risk in the system. Hence, a high value of the multiplicands *OR*, *SR* or *DR* would increase the overall risk for that user.

As a sample implementation, we consider user A, who submits 10 queries at time t as follows: 3 queries

Table 5

Query frequencies for the sample user

Query (q_i)	Role	Frequency	Percentage	OR_{q_i}
q_0	0	48	0.78	1
q_1	0	65	1.05	2
q_2	0	17	0.28	1
q_3	3	1	0.96	1
q_4	3	5	4.81	4
q_5	3	8	7.69	5
q_6	3	19	18.27	7
q_7	3	1	0.96	1
q_8	5	5	3.21	4
q_9	5	19	12.18	6

from role 0, 5 queries from role 3 and 2 queries from role 7. Examining his previous query submissions, we get the following distributions given in Table 5.

In Table 5, we calculate the percentage values for each query by dividing its frequency rate to the total number of queries under that role definition. For example, for q_0, the percentage is calculated as $48/6170 \times 100$. To fill the last column (OR_{q_i}) of Table 5, we use the corresponding ratings for each percentile in Table 1. Finally, the overall occurrence rating is calculated as the sum of *OR*s, which is 32 for the example.

Another important aspect of frequency calculation is how we actually differentiate queries and how their frequencies are distributed among each role. As an example, let's consider the following two queries from role 0:

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

and

```
SELECT *
FROM contract_record
WHERE contract_no = 'm2858';
```

In essence, these two queries are not different from each other, because they are both represented similarly in each of the coarse grain, medium grain and fine grain query representation modes. Under each role in the database, there are several queries with similar representations. For the sake of easiness in processing, we introduce the concept of query template (*QT*) to our scheme. A query template is a prototype that embodies queries with similar representations.

Table 6

Query template frequencies under role 0

QT	Freq.	102	38	100	8	288	4	310	3	139	2
498	623	518	38	336	8	297	4	322	3	141	2
1	327	466	32	84	7	328	4	344	3	143	2
476	298	510	31	98	7	340	4	347	3	145	2
24	288	92	29	162	7	352	4	397	3	147	2
19	282	506	28	516	7	439	4	416	3	151	2
7	254	103	27	22	6	449	4	452	3	153	2
95	246	514	25	107	6	470	4	463	3	154	2
4	217	29	21	338	6	20	3	16	2	155	2
2	203	89	21	522	6	41	3	30	2	156	2
471	172	99	21	43	5	64	3	31	2	157	2
3	171	512	21	52	5	72	3	42	2	159	2
9	163	6	17	82	5	74	3	50	2	160	2
97	152	504	17	199	5	75	3	69	2	161	2
90	134	62	16	214	5	83	3	76	2	163	2
23	130	78	14	286	5	86	3	105	2	165	2
8	127	467	14	502	5	130	3	108	2	166	2
10	127	497	14	508	5	132	3	109	2	167	2
15	116	25	12	14	4	158	3	111	2	173	2
17	116	56	12	27	4	176	3	115	2	174	2
481	106	60	12	37	4	178	3	118	2	175	2
11	88	469	11	58	4	186	3	119	2	177	2
490	70	87	10	63	4	191	3	121	2	179	2
487	60	91	10	70	4	195	3	123	2	180	2
96	58	520	10	88	4	233	3	127	2	181	2
93	52	80	9	101	4	238	3	129	2	183	2
0	48	301	9	172	4	244	3	131	2	187	2
18	43	484	9	184	4	278	3	133	2	188	2
12	42	500	9	249	4	295	3	135	2	...	2 or 1
13	42	48	8	266	4	303	3	137	2	522	1

To reduce the processing load, we analyzed the data set and grouped similar queries under same query templates for each role. This way, 6170 queries in role 0 are reduced to 522 query templates, 4 queries in role 1 generate 4 query templates, 20 queries in role 2 are reduced to 12 query templates, 104 queries in role 3 are reduced to 22 query templates, 1 query in role 4 remains as a single query template, 156 queries in role 5 are reduced to 10 query templates, 10 queries in role 6 are reduced to 2 query templates and finally 1123 queries in role 7 are reduced to 152 query templates.

After we generate the query templates (*QT*), occurrence rating (*OR*) computation becomes a matter of measuring the *QT* frequencies. In Table 6, we list the frequency of each query template (*QT*) under role 0 for our sample data set. We repeated the experiment for all other role definitions and obtained each role's QT frequencies.

According to Table 6, some query templates occur comparatively more frequently than others under role 0; e.g. the query template (*QT*) 498 occurs 623 times, which comprises 10.09% of the 6170 total queries only under role 0. With further analysis, we have also discovered that some query templates occur in other role definition(s), as well (Section 4.3). With that much frequency rate, the queries having the same template with the query template *QT* 498 would have a higher effect on the overall risk calculation. Likewise, the sec-

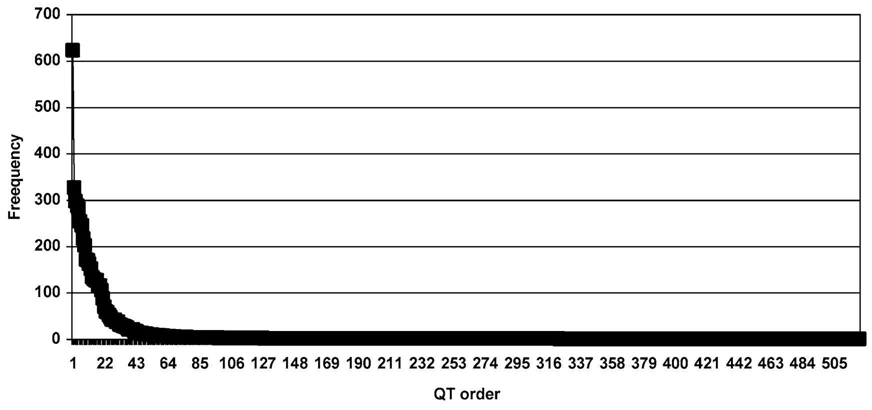

Fig. 4. Chart for QT distributions under role 0.

ond most frequent query template *QT* 1, which is in the form of

```
SELECT *
FROM contract_record
WHERE contract_no = Something;
```

occurs 327 times, which comprises 5.30% of the total queries under role 0, would have a considerable effect on the risk priority number calculation (*RPN*), too. Considering their distributions, *QT* 498 and *QT* 1 have an occurrence rating (*OR*) of 9 each (Table 1).

An interesting observation with Table 6 is that, the frequency distribution of the query templates (*QT*) follows the Zipf's Law [23]: the frequency of an item is inversely proportional to its rank. So, the most frequent item (*QT* 468) occurs approximately twice as often as the second most frequent item (*QT* 1), etc. In Fig. 4, we give the distribution chart of *QT*s under role 0.

4.2. *Severity rating (SR) results*

As we did with occurrence rating (*OR*), we calculate the severity rating (*SR*) similarly: for each user, we process his individual queries q_i from the set of queries (*SQ*) that he has submitted within time period t. We tokenize each q_i to find out the following: (1) what sort of query it is, i.e. either a SELECT, or an UPDATE, or an ADD or a DELETE query; (2) which attribute(s) it is associated with; (3) which relation(s) it is associated with. Then, for each token we perform simple table lookup to assign a predetermined weight index to it.

Let's consider the q_0 of user A:

```
SELECT *
FROM contract_record
```

which lists all attributes from the relation `contract_record` in the database. After tokenizing this statement, we obtain an SQL command SELECT, a relation name `contract_record`, and the whole attributes (*) of the relation `contract_record.` Before we make the weight assignments to each token, we perform a pre-study of all SQL commands together with a pre-study of the relations and their attributes in the dataset. With a thorough analysis, we determine weight values that will be assigned to each SQL command, as well as to each relation and attribute. While doing that, we consider the severity of executing a SQL command and severity of reading, modifying or deleting a relation or an attribute individually. After this analysis is accomplished, the parameters of the system are set and the same weight values can be used repetitively.

Considering our ongoing example, the first token we get for q_0 is the SELECT command and is used to access the database records in a read-only manner. It has a comparatively less severe effect on the database as compared to ADD or UPDATE commands, both of which modify the database. For this reason, the weight assigned to the SELECT command is less than that of ADD and UPDATE commands. With the above mentioned pre-analysis of the set of all SQL commands, we prepared a weight indices table, where each SQL command is assigned a particular weight value. As an example, the weights we assigned for some common SQL commands are as follows: SELECT$\rightarrow$10, ADD$\rightarrow$15, UPDATE$\rightarrow$12 and DELETE$\rightarrow$20. As part of the SQL syntax, the SELECT command is associated with a list of attributes in the database. For the sample query q_0, the use of asterisk (*) indicates all attributes of the relation `contract_record` are involved in the SELECT statement. So, while making weight assignments, we need to consider the whole list of attributes for the relation `contract_record` as: `active_status`, `boc_con_num`, `contract_amount`, `contract_branch`, `contract_date`, `contract_no`, `contract_type`, `customer_id`, `free_type`, `invoice_num`, `is_boc`, `is_member`, `is_new`, `is_sponsor`, `net_sales_amount`, `outstanding_balance`, `post_dated_total`, `referred_media_name`, `remark`, `sales1_id`, `sales2_id` to the calculation of the severity rating (*SR*). There are 21 attributes in the relation contract_record. With an equal weight assignment where each attribute would have a weight of 1, the total weight to be assigned for all attributes of `contract_record` would be 21. But in reality, some attributes have higher severity, which leads to higher weight. For example, the attribute `contract_amount` has a weight of 10, the attribute `invoice_num` has a weight of 4 and the attribute `outstanding_balance` has a weight of 3. So, the total weight for the whole list of attributes in the relation `contract_record` is calculated as $18 + 10 + 4 + 3 = 35$. Therefore, the partial weight value calculated for the first line of query q_0, which is the "`SELECT *`" line, would be $10 \times 35 = 350$.

The second token that contributes to the calculation of the severity rating (*SR*) in the sample query q_0 is the relation name, which is `contract_record`. Among 130 relations in our dataset, we assign a weight index of 24 to `contract_record`. This weight is slightly more than three times the average weight share ($1000/130 = 7.69$) for each relation, if we use a 1000 total scale for weights to be assigned to each relation and had we incorporated an equal weight distribution for relations. But clearly, not every relation has an equal share because some of them store more critical data than the others. The relation `contract_record` is such a relation because it contains the contract information between the patient and the health institute. So, it has higher severity than average relations. This is why we multiply the weight calculation for it by 3. Hence, we calculate the partial weight value for the second line of query q_0, which is the "`FROM contract_record`" line, as $7.69 \times 3 \cong 24$.

Finally, the query risk value for q_0 would be the sum of two lines for the SQL statement: $350 + 24 = 374$. Repeating the risk calculation for the rest of the queries for user A in a similar manner, we obtain a sum of 3480. By looking up Table 2, we find the corresponding severity rating (*SR*) for the value of 3480 and we conclude that the *SR* for user A is 7.

4.3. Detection rating (DR) results

With detection rating (*DR*) we address the problem of coexisting queries under different role definitions. Ideally, roles should have well defined boundaries so that no single query is resubmitted by users that have different roles assigned. Unlike from the other risk components as occurrence rating (*OR*) and severity rating (*SR*), a high detection rating (*DR*) indicates the poor detection capability in the database. With detection capability, we mean the ability of the system to distinguish queries that occur multiple times under various roles. So, for such queries that occur under multiple roles we assign a high *DR* (Table 3). The calculation of *DR* is more complicated than that of *OR* and *SR* calculations: We construct as many groups as the number of queries that belong to distinct roles for an individual user. For the ongoing example, there are 3 such groups because user A submits queries from role definitions 0, 3 and 5. Then, for each group, we consider other users' submissions under the same role definition and construct their clusters by using K-means clustering. By calculating the distance of user A's query submissions under the same role to the already formed clusters, we detect how far is user A (actually his queries) from the other users (actually from their queries) under this particular role definition. If the distance is high.

By analyzing queries under eight different role definitions, we obtained probability values that are not very distinctive. This is different from what we had anticipated. By further analyzing the queries under each role, we realized that the same or similar queries occur in multiple role definitions.

Table 7 lists the number of queries that are common under each role definition. In this table, the value in cell (i, j) denotes the count of queries that are submitted by users under roles i and j. As seen from the table, 521 queries occur both under role 0 and under role 7.

Table 7
Common query occurrences under different roles

	Role 0	Role 1	Role 2	Role 3	Role 4	Role 5	Role 6	Role 7
Role 0		4	14	2	0	1	0	521
Role 1			4	0	0	0	0	4
Role 2				1	0	1	0	19
Role 3					1	8	4	7
Role 4						1	1	1
Role 5							10	13
Role 6								4
Role 7								

This is a very high proportion: For role 0, 8.44% of its queries are common with queries under role 7, and when we look from the other side the situation looks even worse: 46.39% of the queries under role 7 are common with queries under role 0.

Unfortunately, the occurrence of overlapping queries is far from what we call the ideal case. Normally, we expect role definitions to have strict boundaries, i.e. queries under each role definition to be completely different from that of other role definitions. In this way only, queries can be clustered properly and the ones that are distant from a group of clusters under the same role definition are considered to have higher risk than the others. But this overlapping among different role definitions causes variation from correct probability values.

What we suggest is to use query templates instead of individual queries under limited role definitions. In this fashion, users are limited to obey the query templates and cannot submit queries upon their requests. This helps eliminate the probability of unanticipated queries for each role definition and hence alleviate the likelihood of high risk values. In order to detect the query templates in the current database, we analyzed queries under each of the eight role definitions and listed the query templates consisting of similar queries. This yielded the chart below:

The uneven distribution of query templates among roles in Fig. 5 is a consequence of the uneven query distribution among roles (Table 4).

To calculate the detection rating (*DR*) value for user A, we repeated three different experiments for roles 0, 3 and 5 since the set SQ contains query submissions under these role definitions. We give the results of our experiments on role 0 in Fig. 5. We picked 99 users under role 0 and listed their query submissions by using the query templates (*QT*). As for the last user, we listed user A's two queries (in the form of QT again). Then, we applied K-means clustering to first cluster the existing users' queries (actually the QTs) and then to calculate the distance (d_q) of user A to these clusters. In this manner, we detect whether user A behaves abnormally as compared to the other users under role 0 or not.

For more precise results, we used normalized QT values in Table 8. For role 0, we calculated the *distMeasure* as 2.78, so the detection rating $DR = 9$ for queries under role 0. Repeating the experiments, we calculated the DR values as 4 and 6 under roles 3 and 5, respectively. Hence, the total DR for user A is $(9 + 4 + 6 = 19)$.

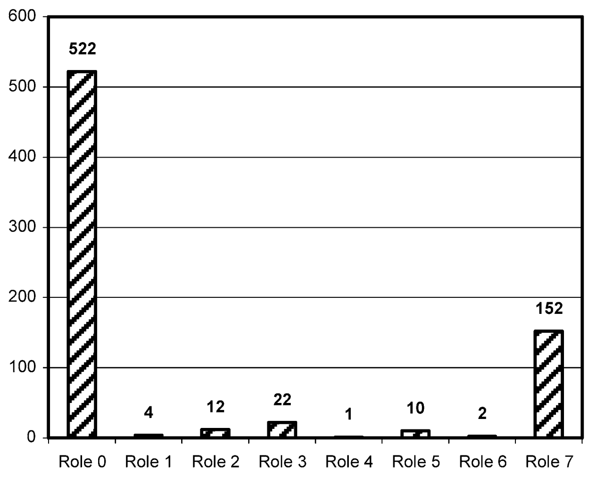

Fig. 5. Query template distribution among roles.

Table 8

Synthetic data view for detection of malicious user

	QT1	QT2	QT3	QT4	···	QT50
User X	162	141	263	34	···	5
User Y	206	49	0	202	···	
	...	...	...	...	...	...
User A						

To calculate the overall risk value (*RPN*) for user A, we combine the risk priority number parameters as occurrence rating (*OR*), severity rating (*SR*) and detection rating (*DR*) to get: $RPN = 32 \times 7 \times 19 = 4256$.

As part of the pre-analysis of the dataset, we need to determine what level of RPN should be considered as a high risk and what level as low risk. In our sample dataset, we assume that the highest value of RPN could be $100 \times 100 \times 100 = 10^6$. So, the value of $RPN = 4256$ calculated for user A is considerably low level of risk. As part of the risk mitigation and decision making step (Section 3.4), we give the database manager these options as: to grant further access to the database for user A; to reject his requests for further access, because he did not have a very low level of risk; to log his activities because he has a certain risk level assigned and he is suspicious now; or to encrypt the records in the database so that no further user would be able to access them easily. This decision depends on the strictness of the database and the security policy that is chosen by the database administration.

4.4. Discussion of the methodology

The general risk formula used in Eq. (3) is actually in parallel with the utility function (Eq. (5)) of the decision theory:

$$\sum_{1}^{n} p_i u(x_i). \tag{5}$$

As risk formula represents the overall risk as the product of probability of each risk factor and the consequence of these risk factors; maximum utility function computes maximum utility as the product of each participant involved in utility calculation, and its weight [14,20]. In that sense, both risk formula and utility function are general formulas to be used in multidisciplinary research areas. One such study belongs to Sedeno et al. [17]. In this study, authors measure the weight of several factors to compute a job satisfaction value. So, our study incorporates a universal methodology for risk calculation.

The general formula for the calculation of risk (Eq. (3)) is simple and concise. By definition, this calculation involves probability of the occurrence of a consequence. The key point in this definition is that, the consequence has not happened yet. Hence, risk is associated with uncertainty. FMEA scheme, relying on the calculation of risk via above mentioned consequence factor also involves uncertainty. Still, one can generate a well representing model for the not-occurred events by a through observation and a comprehensive modeling.

5. Conclusion and future work

This study proposes a novel risk management system to provide enhanced security in access control for RBAC employed distributed databases. The system introduced is made up of two components as risk analysis and risk control. Within our design, we consider risks incurred by illegitimate credentials, role misuse/abuse and/or system failure to the system. We base our risk measurement on user queries for which we use a standard three layered representation as coarse grain, medium grain and fine grain. Experiments showed us that our scheme can detect the risk in user queries.

For risk analysis, we employ a three parameter risk evaluation scheme that involves query history, analysis of the current query and detection rating.

We used a real dataset in our experiments and obtained promising results. With ongoing work, we are planning to employ other clustering algorithms (e.g. EM algorithm) in our risk model. Furthermore, we are searching other real world datasets in which role borders are well defined and queries do not overlap to run our scheme.

References

[1] B. Ayyub, *Risk Analysis in Engineering and Economics*, Chapman & Hall, CRC, USA, 2003.

[2] J. Bacon, N. Dimmock, D. Ingram, K. Moody, B. Shand and A. Twigg, SECURE Deliverable 3.1: Definition of risk model, December 2002.

[3] E. Bertino, A. Kamra, E. Terzi and A. Vakali, Intrusion detection in RBAC-administered databases, in: *21st Annual Computer Security Applications Conference*, Tucson, AZ, USA, December 2005.

[4] C. Borgelt and R. Kruse, *Graphical Models Methods for Data Analysis and Mining*, Wiley, Chichester, UK, 2002.

[5] V. Cahill, W. Wagealla, P. Nixon, S. Terzis, H. Lowe and A. McGettrick, Using trust for secure collaboration in uncertain environments, *IEEE Pervasive Computing* **2** (2003), 52–61.

[6] M. Carbone, N. Dimmock, K. Krukow and M. Nielsen, Revised Computational Trust Model, EU IST-FET Project Deliverable, 2004.

[7] M. Carbone, M. Nielsen and V. Sassone, A formal model for trust in dynamic networks, in: *1st IEEE International Conference on Software Engineering & Formal Methods*, Brisbane, Australia, September 25–26, 2003.

[8] N. Dimmock, How much is enough? Risk in trust-based access control, in: *IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises – Enterprise Security*, Linz, Austria, June 2003, pp. 281–282.

[9] N. Dimmock, J. Bacon, D. Ingram and K. Moody, Risk models for trust-based access control (TBAC), in: *Third International Conference iTrust 2005*, Paris, France, May 23–26, 2005.

[10] N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon and K. Moody, Using trust and risk in role-based access control policies, in: *9th ACM Symposium on Access Control Models and Technologies*, Yorktown Heights, NY, USA, June 2–4, 2004.

[11] C. English, W. Wagealla, P. Nixon, S. Terzis, H. Lowe and A. McGettrick, Trusting collaboration in global computing systems, in: *Trust Management: First International Conference, iTrust 2003*, Heraklion, Crete, Greece, May 28–30, 2003, LNCS, Vol. 2692, Springer-Verlag, 2003, pp. 136–149.

[12] D. Ferraiolo and R. Kuhn, Role-based access control, in: *15th NIST-NSCS National Computer Security Conference*, Baltimore, MD, USA, 1992, pp. 554–563.

[13] M.P. Gallaher, A.C. O'Connor and B. Kropp, The economic impact of role-based access control, Planning Report 02-1 for NIST, NC, USA, March 2002.

[14] B.R. Munier and C. Tapiero, Risk attitudes, in: *Encyclopedia of Quantitative Risk Assessment and Analysis*, Wiley, New York, 2008.

[15] N. Nissanke and E.J. Khayat, Risk based security analysis of permissions in RBAC, in: *2nd International Workshop on Security In Information Systems*, Porto, Portugal, April 2004.

[16] R.S. Sandhu, E.J. Coyne, H.L. Feinstein and C.E. Youman, Role based access control models, *IEEE Computer* **29**(2) (1996), 38–47.

[17] M.G. Sedeno, M.I.B. Garcia and C.G. Tejera, The function of subjective utility as an indicator of job satisfaction, *Psychology in Spain* **4**(1) (2000), 129–138.

[18] B. Shand, N. Dimmock and J. Bacon, Trust for ubiquitous, transparent collaboration, *Wireless Networks* **10** (2004) 711–721.

[19] P.N. Tan, M. Steinbach and V. Kumar, *Introduction to Data Mining*, Pearson Education, USA, 2006.

[20] C. Tapiero, Risk management: An interdisciplinary framework, ESSEC Research Center, France, 2003.

[21] Q. Yao, A. An and X. Huang, Finding and analyzing database user sessions, in: *10th International Conference Database Systems for Advanced Applications*, Beijing, China, April 17–20, 2005, LNCS, Vol. 3453, 2005, pp. 851–862.

[22] C.N. Zhang and C. Yang, An object-oriented RBAC model for distributed system, in: *Working IEEE/IFIP Conference on Software Architecture (WISCA'01)*, Amsterdam, The Netherlands, August 28–31, 2001, pp. 24–32.

[23] G.K. Zipf, *Selected Studies of the Principle of Relative Frequency in Language*, Harvard University Press, Cambridge, MA, USA, 1932.