



LABORATOIRE



INFORMATIQUE, SIGNAUX ET SYSTÈMES
DE SOPHIA ANTIPOLIS
UMR 6070

RBAC+ : ENFORCING ACCESS CONTROL FOR RBAC-ADMINISTERED WEB DATABASES

Ahlem BOUCHAHDA-BEN TEKAYA

Equipe KEWI

Rapport de recherche
ISRN I3S/RR-2009-18-FR

Decembre 2009

RÉSUMÉ :

Dans les applications Web, la nécessité de la médiation toutes les interactions de base de données par l'intermédiaire du serveur d'applications, masque l'identité de l'utilisateur final au système de gestion de base de données (SGBD). Le SGBD n'a aucune connaissance de l'utilisateur de l'application Web, mais uniquement de l'application Web elle-même. Ainsi, toutes les décisions d'autorisation sont fondées sur le compte sous lequel l'application Web s'exécute. Ainsi, il n'y a pas de contrôle d'accès réel pour les utilisateurs finaux, car BD ne peut les identifier et ainsi, il ne peut pas leur fournir des autorisations appropriées. Il est évident que, les approches actuelles de contrôle d'accès aux bases de données ne correspondent pas à des applications web car elles sont principalement fondées sur des identités d'utilisateurs individuels. Pour combler cette lacune de sécurité une renforce du système de contrôle d'accès est nécessaire. Dans ce papier, nous proposons (RBAC +), une extension du standard NIST RBAC (Role-Based Access Control) avec les notions d'application, profil d'application et les sous-sessions d'application afin de distinguer les utilisateurs finaux qui exécutent la même application et ainsi de leur fournir seulement les rôles nécessaires. Ceci permet d'arrêter les accès involontaires aux données dans une application. Enfin, un attaquant qui brise les mécanismes de sécurité à la couche serveur d'applications ne peut plus échapper aux mécanismes de sécurité renforcés à la couche de SGBD. Le mécanisme de contrôle d'accès proposé accroît la capacité de détection des opérations malveillantes, la cause dominante qui démolit la sécurité du système de base de données. Par conséquent, les attaques causées par des opérations malveillantes peuvent être détectés et annulés en temps opportun. Le mécanisme proposé permet de détecter et filtrer même des types d'attaques plus complexes telles que les violations de la logique de métier.

MOTS CLÉS :

Contrôle d'accès au SGBD, profils des applications web, détection d'attaque

ABSTRACT:

In web-based applications, the necessity of mediating all database interaction through the application server, masks the identity of the end user to the Database Management System (DBMS). It has no knowledge of the web application user but only of the web application itself. Thus, all authorization decisions are based upon the account under which the web application executes. Thus, a web application, in performing a database transaction on behalf of a web application user, effectively delegates its full access privileges to that user and hence, there is no real access control for the real users because it cannot identify them and so, it cannot supply them with proper authorizations. Hence, current approaches to access control on databases do not fit web databases because they are mostly based on individual user identities. To fill this security gap, the definition of application aware access control systems is needed. In this paper, we propose (RBAC+), an extension of the NIST RBAC (Role-Based Access Control) standard with the notions of application, application profile and sub-application session to distinguish end users that execute the same application and providing them by only the needed roles. This closes unintended data access channels in the application. Moreover, an attacker who breaks down the security mechanisms at the application server layer cannot escape the stronger security mechanisms at the DBMS layer. The proposed access control mechanism enhances the ability of detecting malicious transactions, the dominant cause that demolishes database system, by tracking application users thought a whole session. Hence, attacks caused by malicious transactions can be detected and canceled timely before they succeed. Even more complex kinds of attacks such as business logic violations may be detected and filtered.

KEY WORDS :

DBMS Access control, web application profiles, attack detection

***RBAC*⁺ : Enforcing Access Control for RBAC-administered Web Databases**

Ahlem BOUCHAHDA-BEN TEKAYA^{1,2}

¹ Laboratoire I3S, (Université de Nice-Sophia Antipolis / CNRS)
Bâtiment Polytech'Sophia - SI 930 route des colles

B.P. 145 F-06903 Sophia-Antipolis Cedex

² Equipe de recherche sécurité numérique

Ecole supérieure des communications de Tunis

Cité Technologique des Communications Rte de Raoued Km 3.5 2083, Ariana Tunisie

Abstract. In web-based applications, the necessity of mediating all database interaction through the application server, masks the identity of the end user to the DataBase Management System (DBMS). It has no knowledge of the web application user but only of the web application itself. Thus, all authorization decisions are based upon the account under which the web application executes. Thus, a web application, in performing a database transaction on behalf of a web application user, effectively delegates its full access privileges to that user and hence, there is no real access control for the real users because it can not identify them and so, it cannot supply them with proper authorizations. Hence, current approaches to access control on databases do not fit web databases because they are mostly based on individual user identities.

To fill this security gap, the definition of application aware access control systems is needed. In this paper, we propose (*RBAC*⁺), an extension of the NIST RBAC (Role-Based Access Control) standard with the notions of application, application profile and sub-application session to distinguish end users that execute the same application and providing them by only the needed roles. This closes unintended data access channels in the application. Moreover, an attacker who breaks down the security mechanisms at the application server layer can not escape the stronger security mechanisms at the DBMS layer. The proposed access control mechanism enhances the ability of detecting malicious transactions, the dominant cause that demolishes database system, by tracking application users through a whole session. Hence, attacks caused by malicious transactions can be detected and canceled timely before they succeed. Even more complex kinds of attacks such as business logic violations may be detected and filtered.

1 Introduction and Motivation

Web applications are extremely popular today, significantly affecting all aspects of our lives, due to the ubiquity of web browsers, and the convenience of using a web browser as a client. The ability to update and maintain web applications

without distributing and installing software on potentially thousands of client computers is a key reason for their popularity.

These applications have direct access to back-end databases that are called Web Databases and often contain confidential, or even sensitive, information, such as customer and financial records. This information, if compromised, can have a very serious impact on both the organizations that deploy them and on the users who access them. Thus, protecting data stored in web databases has become a strong need.

Access control is the primary means of attack prevention for databases. But as long as web databases cannot identify their real users, they cannot supply them with proper authorization. Database Views are another means of unauthorized access restriction as they can define the only part of a database relevant to a user. But for web databases they are useless since the only recognized user of a database is the user of the application server.

In fact, web applications are run from the user's browser windows. The browser does not directly connect to the database, but instead transfer a request to a web server who processes the request and if an access to the underlying database is needed, transfers it to the application server which performs a transaction to the database. The result of this three (or more)-tier architecture, is that the database does not identify the user who accesses it. From the database point of view, the user accessing the database is the user of the application server named *DataBase User* (DBU) and not the real user called indifferently *Application User* or web user. Even, if an identification process is required by an application, users of the application perform the process of identification to the application and not to the database. It is the application's responsibility to decide who the user who tries to connect is.

The situation described above implies that traditional identity-based mechanisms for performing access control are useless for web databases. Further, a DBMS can not handle users who access it indirectly via the application server, no user-based access control can be applied since the only recognized user is the user of the application server and for most of the web applications this is the user with very high privileges.

Databases can no longer differentiate between transactions of different application users. The principle of minimal privilege is violated. It is impossible to authorize the web application user with appropriate privileges at the database level: all application users have access to the same data. Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to view sensitive data, or use unauthorized functions. So, no more fine-grained access control to the database exists and authorization can be provided only at the application level.

Therefore, web applications are exposed to many illegal accesses and attacks that are very hard to prevent and detect. Besides the famous SQL injection attack that was studied extensively (A state of the art is given in [13]), there is one interesting kind of attacks, the Business Logic Violation attack for which satisfactory solutions are still lacking.

Preventing illegal accesses to the database at the application server layer is not effective, as it try to protect the vulnerable database by some middleware ad hoc methods. Instead, we suggest dealing with the root of the problem and providing a new method to protect databases by means of proper access control mechanisms that will be implemented by the databases themselves.

Hence, an access control model with application awareness is needed since application is the missing piece in the access control process. In this paper, we suggest an access control model that we call $RBAC^+$. This model extends the RBAC model (Role-Based Access Control Model)[14] with the concepts of *application*, *application profile* and *sub-application session*. We choose RBAC since it has been widely deployed in various commercial DBMS products. Furthermore, it has been standardized in 2004 [14]. This implies that an access control model based on RBAC, could be easily deployed in practice. What is important is that the $RBAC^+$ model proposed in this paper tightly integrates and builds upon the standard ANSI RBAC model [14]. Thus, it could be realized with only relatively few changes to the standard.

Our approach focuses on detection and prevention of malicious transactions. Malicious transactions are transactions that access database without authorization, or transactions that are issued by users who are authorized but abuse their privileges. The $RBAC^+$ monitors transactions issued by users and malicious transactions are viewed as intrusion behaviors. If a malicious transaction is identified, the $RBAC^+$ cancel the transaction before it succeeds, thus minimize damage caused by malicious transactions.

1.1 Contribution

In this work, we address a complex information system infrastructure consisting of multiple layers where several applications may access a single database, with numerous users at both the application and database layer, including persons, application users, and database users. If users and/or applications share accounts, how can accesses to the database be correlated to users? As accesses to the data occur through several layers, starting with a person then the application, which, in turn, performs operations on the database, correlating anomalous behaviors with a person is not a trivial task. To address these problems, we propose $RBAC^+$, an extension of RBAC, able to detect malicious users and stopping the attack before it succeeds.

Assuming that the database management system (DBMS) has an RBAC model in place, the key idea of our approach is as follows. We create application profiles that represent all the possible execution paths of the application. An execution path is a sequence of SQL queries and each SQL query is represented by a set of permissions necessary for its execution. The program code can be used for this purpose.

Given the permissions necessary to the execution of an application and the set of roles that the underlying database user (DBU) is authorized for, we calculate for each pair (application, DBU) the subset of roles to activate in a web user session, called sub-application session. It is called so because, in the context of

a web application, a web user session is included in a database session. Hence, a sub-application session, contains only the permissions really needed to fulfill exactly the tasks it was created for, and so we take advantage of all RBAC assets such as least privilege and separation of duty that make of it the most widely accepted as the proven technology for access control.

A sub-application session allows to the DBMS distinguishing between web users working with the database, thus solving the first major problem of fine-grained authorization at the database level. It will also allow distinguishing between the requests of different web users that belong to the same database session, thus solving the second problem of user-session's traceability for web applications.

When the web user logs in, the SQL queries that he submits are associated with a database session, an application and the underlying database user that issued them. All queries belonging to a sub-application session must match an application execution path else the access is denied because the action to be executed is illegitimate. Now, when an employee wants to attack enterprise resources, he, for example, can submit an SQL injection attack. But because his database privileges are limited only to legitimate actions, an SQL injection will be entirely mitigated or at least, its effect is strongly limited. The importance of our solution is that it enforces access control based on business application logic rather than primitive reads and writes. A user's ability to access and manipulate data is typically dependent of the application function the user executes thus reducing drastically attacks against databases and in particular, business logic violation attacks because an action may be legitimate on its own but illegitimate in the context of a whole session. Take the example of an online retailer application. The process of placing an order involves the following stages : (1) Browse the product catalog and add items to the shopping basket; (2) Return to the shopping basket and finalize the order; (3) submit credit card details; (4) Enter delivery information;

If the intruder can submit the insert statement into the Orders table without submitting an insert into the Credit Card table, then he can buy goods without paying. In such a case the session of the intruder will have an insert into the Orders table without an insert into the Credit Card table and this business rule violation can be detected only at the session level since each statement by itself is a legitimate one. Databases cannot prevent them because the existing database access control can grant or revoke access to resources only according to the accessor identity/role. It cannot rely on the business logic of an organization. The developers assume that users would always proceed to the functionality as they expected and access the stages in the intended sequence, because this was the order in which the stages are delivered to the user by the navigational links and forms presented to their browser. However, users control every request they make to the application and so can access any stage of the ordering process in any sequence. By proceeding directly from stage 2 to stage 4, an attacker could generate an order that was finalized for delivery but that had not been actually paid for.

When multistage functions are accessed out of the expected sequence, it is com-

mon to encounter a variety of anomalous conditions within the application, such as variables with null or uninitialized values, a partially defined or inconsistent state, and other unpredictable behavior. In this situation, the application may return interesting error message and debug output, which can be used to better understand its internal working and thereby fine-tune the current or a different attack. Sometimes, the application may get into a state entirely unanticipated by developers, which may lead to serious security flaws [12].

Many other web database attacks are very specific to the business/application logic. We track users at the session level, so we are able to prevent attacks such as the business logic violation, which cannot be seen at the statement/transaction level, as their effect accumulates during an entire session.

1.2 Paper Road map

The rest of the paper is organized as follows. We present the related work and discuss it in Section 2. In section 3, we present the background concepts necessary for the understanding of the rest of the paper. We define formally and detail our model in Section 4. In Section 5, we discuss the side effects on other RBAC Components. We conclude our work and present future work in Section 6.

2 Related Work and Discussion

To the best of our knowledge, little existing work has addressed such an authorization problem for web databases. To protect web databases from attacks of malicious users, two main approaches exist. The first consist on using ad hoc tools specifically oriented to the detection of specific kinds of attacks like SQL injection [13]. The second consists on using Intrusion Detection Systems (IDSs). Intrusion detection approaches can be divided into misuse detection and anomaly detection. Misuse detection techniques use patterns of well-known attacks to match and identify known intrusions. They perform a pattern matching between the current behavior captured and attack signature. If the matching succeeds, then the system generates an alarm. This approach can detect known attacks accurately, but is ineffective against previously unseen attacks, as no signatures are available for such attacks.

Anomaly detection overcomes the limitation of misuse detection by focusing on normal system behaviors, rather than attack behaviors. The profiles of normal system behaviors are usually created by machine learning techniques. Any deviation from normal system behaviors is treated as a potential attack.

Among the most important database intrusion detection systems proposed in the literature, we cite DIDAFIT [3] which presents an approach for detecting illegitimate database accesses by fingerprinting transactions. The main contribution of this work is a technique to summarize SQL statements into compact regular expression fingerprints. The system detects an intrusion by matching new SQL statements against a known set of legitimate database transaction fingerprints. Also, Chung et al. proposed DEMIDS [6], an IDS tailored for relational database

systems, it uses audit log data to derive profiles describing typical patterns of accesses by database users. The approach assumes that the access pattern of users typically forms a working scope which comprises sets of attributes that are usually referenced together with some values. DEMIDS assumes domain knowledge about the data structures and semantics encoded in a given database schema. Distance measures are then used to guide the search for frequent itemsets describing the working scope of users and hence users behaviors.

Bertino et al. [5] presents an anomaly detector for RBAC-administered databases. The proposed approach builds profiles using syntactic information from SQL queries appearing in the database log to model normal user actions and use it to identify intrusions.

Lee et al. [7] proposed a real-time database intrusion detection using time signatures. If a transaction attempts to update a temporal data which has already been updated in that period, an alarm is raised.

Valeur et al. present an anomaly based IDS for the detection of SQL based attacks exploiting vulnerabilities in web-based applications to compromise a back-end database. The IDS taps into the communication channel between web-based applications and the database server. SQL queries performed by the applications are intercepted and sent to the IDS for analysis. The drawback of this approach is that it is limited to detecting only three specific classes of SQL-based attacks. The IDS proposed by spalka et al. [1] is tailored to relational databases and consists of three components: an anomaly detector for the database, an anomaly detector for the user interaction, and a misuse detector for the database scheme.

The approach proposed by Hu et al. [4] determines dependency among data items where data dependency refers to the access correlations among data items. These data dependencies are generated in the form of classification rules, i.e., before one data item is updated in the database, which other data items probably need to be read and after this data item is updated, which other data items are most likely to be updated by the same transactions. Transactions that do not follow any of the mined data dependency rules are marked as malicious transactions.

Barbara et al. [10] use hidden markov model (HMM) and time series to determine malicious data corruption. They build a database behavioral model using HMM that captures the changing behavior over time. Malicious patterns are the ones that are not recognized by the HMM with high probability.

The approach by Zhong et al. [9] consists of an algorithm to mine user profiles based on the queries submitted by the user.

Wenhui et al. [11] present a two layer mechanism to detect intrusions in web-based database systems. The first layer is used to build historical profiles and pre-alarms may be generated due to abnormal behavior. The second layer further verifies the alarms generated by the layer one.

Although we believe that database IDSs can perform very well in detecting anomalous behaviors and that IDS should play an important role in database security, we have to point out that the web application's access to databases remains untraceable. Further, an IDS can not overcome the absence of web database internal access control and the uselessness of views as a means of ac-

cess restriction.

Moreover, with the assumption that the attack does not go unnoticed, IDSs focus on detecting attacks after the malicious user has accessed the DB with all the damage it could cause. However, it is not always the case because IDSs, in practice, when profiling normal activities for anomaly detection purposes, it is only a subset of normal activities which is profiled since the transactions learning depends on the utilization profile of the database. In many cases, large database applications include functionalities that are only executed from time to time, for example at the end of the week or end of the month. Thus, we have a coverage problem since only frequently used functionalities which are profiled which explains in part the high rate of false positives of anomaly-based intrusion detection.

Our solution to this problem is to profile user behavior based on the application logic. In fact, each application have a way of working to accomplish its features following ordered actions. Besides, in a typical web environment, transactions are programmed at the application level, which means that the set of transactions remains stable, as long as the application is not changed. For example, in a banking database application users can only perform the operations available at the application interface (e.g., withdraw money, balance check account, etc). No other operation is available for the end-users (e.g., end-users cannot execute ad-hoc SQL commands). This way, it is possible to profile application features and thus reduced risk of false alarms.

What we propose is strengthening access control and continuously monitor users. Consequently, the majority of attacks can be stopped from the access control stage and the IDS will be used to detect attacks that have escaped the access control stage. Intrusion detection without enforcing access control is not as efficient and effective. IDS is a complement but can not, alone, protect DB from attacks.

Our approach is based on the intuition that the web application code implicitly contains a policy that allows for distinguishing legitimate and malicious queries. So, the source code contains enough information to infer models of the expected, legitimate sequences of SQL queries generated by the application. We choose to model the application behavior rather than the user behavior because a system based on modeling user profiles results in a large false alarm rate. This is because of two reasons. First, the user behavior is not fixed and changes overtime and, second, profile based systems employ threshold to determine the acceptable deviation in normal activities. The thresholds are often determined empirically and, hence, may be unreliable.

3 Background and Terminology

In this section, we present the necessary background on web applications, web databases and the RBAC model.

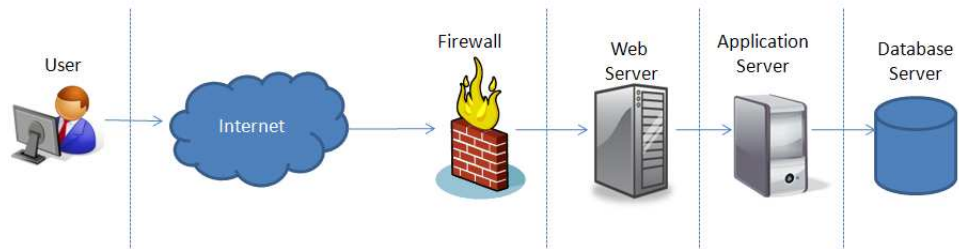


Fig. 1. Architecture of a typical web application

3.1 Web Application Overview

Web applications are the business logic that enables user's interaction with the website, and the transacting and interfacing with all the back-end data systems. Examples include applications that allow users to look up their account information at their bank and move funds, applications that allow users to buy things online, etc...

Web applications commonly use a combination of server-side script (ASP, PHP, etc) and client-side script (HTML, Javascript, etc.) to develop the application. The client-side script deals with the presentation of the information while the server-side script deals with the essential functional features and especially database interactions for accessing internal data and performing transactions. that we are interested in to build the application profile.

A web application has a distributed n-tiered architecture. Typically, there is a client (web browser), a web server, an application server (or several application servers), and a database server. Figure 1 presents a simplified view of a web application. There may be a firewall between web client and web server to protect it from intrusions and allow traffic at port 80 only.

The most common web application architecture uses a three-layered model :

- The *presentation layer* is the front-end of the web application. Users interact with this layer by opening a web browser, the application interface, and typing a URL. The browser requests the resource by sending an HTTP request to the web server which hosts the business layer. Web pages are delivered to clients in response to requests for URLs.
- The *business layer* receives requests from users and processes them and passes the dynamic part to the application server, which in turn, translates these requests to database SQL commands and, using the results of those commands, generates the response that is sent back to the browser for final

presentation to the user. This layer interacts with the next layer the data layer, through SQL statements.

- The *data layer* is the repository of the website information. It receives requests from the application server for access to information in the database server through data access technologies such as ODBC or JDBC. The application server connects to the database with one database user (or with a very limited number of DB users) that acts as an agent connecting and accessing the database on behalf of the application users. So that many application users are mapped to one database user as shown in Figure 2.

The above architecture has some security problems:

- All inputs sanitizations and validations can be easily bypassed since they are on the client side.
- The business layer serves many requests from different users simultaneously. Thus the business layer must be aware of each user session and may be prone to session hijacking attacks.
- The data layer is independent of the above layers, thus it is desirable that it will support its own authorization mechanisms.

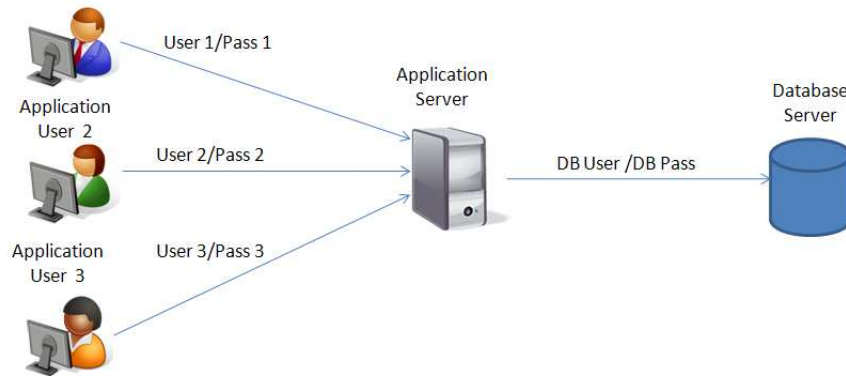


Fig. 2. Mapping of Application users to DB user

If an identification (authentication) process is required by an application, users of the application perform the process of identification to the application and not to the database. It is the application's responsibility to decide who the user who tries to connect is. But the application can be assisted by the database to complete the identification process. In such a case, the application can check the user's identity in a database that will be called the Users Repository. For some applications the users repository and the application database are the same, but they can be different databases. The important thing is that using the

mechanism of distributed query, an application database can access the users repository.

3.2 Application Session vs. DB Session

We use the term "application session" to denote all the transactions that an application runs on behalf of a user from the moment he connects to the application until the moment he disconnects. An application session can run transactions on different database connections because of connection pooling issues. As a result, many application sessions can run transactions on the same connection to the database and the same application session can run different transactions on different database connections.

3.3 Role-Based Access Control

The RBAC model, as defined in the NIST standard, consists of four basic sets of elements: *users*, *roles*, *permissions* and *sessions* as shown in Figure 3.

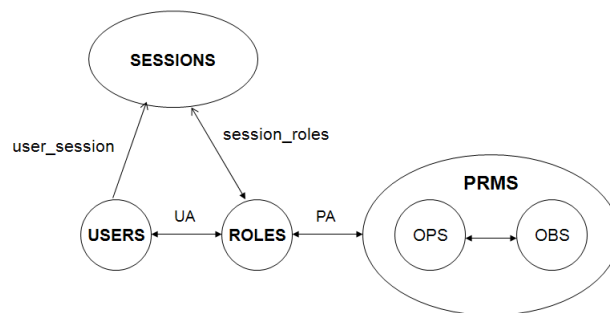


Fig. 3. Core RBAC

- *User* is as a human being or an autonomous agent.
- *Role* represents the function of a user within an organization. A role confers a set of permissions on the user.
- *Permission* is an approval to perform an operation on one or more objects. An object is a resource that shall be protected. The types of operations and objects in a database management system include insert, delete, select and update.
- *Session*, When the user logs in, a session is established during which the user activates some subset of roles that he or she is assigned. The permissions available to the user of the session are thus the permissions assigned to the roles that are currently active across all the users sessions.

Over the above sets of elements, a number of relations are defined. Sets and relations as a whole constitute the basic RBAC layer, the *Core RBAC* defined as follows:

Definition 1. (*Core RBAC*): *Core RBAC* consists of the following components:

- The sets U , R , $PRMS$ and SES represent the set of users, roles, permissions and sessions. We define:
 - $UA \subseteq U \times R$. The user-assignment relation that assigns users to roles.
 - Assigned User: $R \rightarrow 2^U$. The mapping from a role to a set of users.
 - $PA \subseteq R \times PRMS$. The permission-assignment relation that assigns permissions to roles.
 - Prms Assignment: $R \rightarrow 2^{PRMS}$. The mapping of a role into a set of permissions.
 - Session User: $SES \rightarrow U$. The mapping from a session to a user.
 - Session Role: $SES \rightarrow 2^R$. The mapping from a session to a set of roles.

Core RBAC defines the minimum collection of RBAC elements, element sets and relations for a RBAC system to be defined. Besides the Core RBAC, the model consists of additional components that can optionally be used: *Hierarchical RBAC* and *Constrained RBAC*.

The Hierarchical RBAC component adds relations for supporting role hierarchies. A hierarchy is a partial order defining a seniority relation between roles, whereby senior roles acquire the permissions of their juniors. Constrained RBAC adds Separation of Duty (SoD) constraints to the RBAC [15].

SoD constraints are in general used to prevent conflicts of interest in an organisation possibly arising because of the usage of conflicting roles. Roles are conflicting if a user can gain the authorisation from them to exercise conflicting operations.

Two different SoD constraints are considered: the static and the dynamic one. Static constraints prevent the assignment of conflicting roles to users, so that a user assigned to a given role can never be assigned to a role conflicting with the first one; in the dynamic case, the constraints enforce the activation of conflicting roles within a user session so that a user cannot have conflicting roles active at the same time.

4 The $RBAC^+$ Core Model

The central idea of $RBAC^+$ is including the concepts of application, application profile and sub-application session when controlling the access to web databases. The application profile is necessary to track the user behavior throughout a whole session and mainly to prevent business logic violation attacks from the access control phase. Such attacks compromise the business logic and can be seen only at the session level. Databases cannot prevent them because the existing database access control can grant or revoke access to resources only according to the accessor identity/role. It cannot rely on the business logic of an organization.

Thus the database’s access control is useless in such a case and business logic violations remain unprevented. To the best of our knowledge, there is no approach proposed that prevents such attacks, the approaches treating this type of attack uses IDSs to detect and there is no way to prevent. Thus, we must enforce the database access control mechanism to be able to prevent such logic violation attacks. Consequently, the access control system must learn the business logic of the web application which is represented by the application profile and any user session must correspond to one path in the application profile graph, else it can be considered as intrusive. There are two ways to building application profiles. The first consists on analyzing the application code and the second consists on analyzing logs.

We choose the first alternative. In fact, a program in a web application normally interacts with database through statements which execute the SQL data manipulation language (DML) operations such as select, insert, update or delete.

An application profile gives all the possible execution paths (sequences of selects, inserts, updates, and deletes) of database interactions. Each path consists of a sequence of SQL statements that are related to each other in terms of the business application logic. Each statement represents a specific unit of work that the application needs to do in order to execute its function.

We, now, introduce a rigorous definition of the model. The purpose is to provide a comprehensive definition of the components, thus including all the aspects of the model.

For sake of readability, in what follows we present the model as organized in a number of logical parts, one for each major set of the RBAC model, that is roles, permissions, users, sessions. The general structure of the model is illustrated in Figure 4. We use the graphical representation adopted in RBAC. In particular, APPS, AP and SASES represent the sets of applications, application profiles and sub-application session respectively.

4.1 Application Profile

An Application Profile is a directed graph that consists of a set of N nodes and a set $E \subseteq N \times N$ of directed edges between nodes. The edges represent the valid execution sequences of an application APP. Each node represents an SQL statement. The graph has one start node and one or many end nodes where the application execution starts and finishes, respectively. An execution path may include cycles representing the repetitive execution of sets of commands (a typical example of cycles in a transaction is the insertion of a variable number of lines in a customer’s order).

Definition 2. Application Profile. An application profile $AP = (N, E)$ where:

- $n \in N$ is a binary vector with the length equal to the number of permissions in the application APP, where the i^{th} bit is 1 if the SQL query needs the permission p_i to be executed, else bit i is 0. $p_i \in PRMS$.
- $E \subseteq N \times N$.

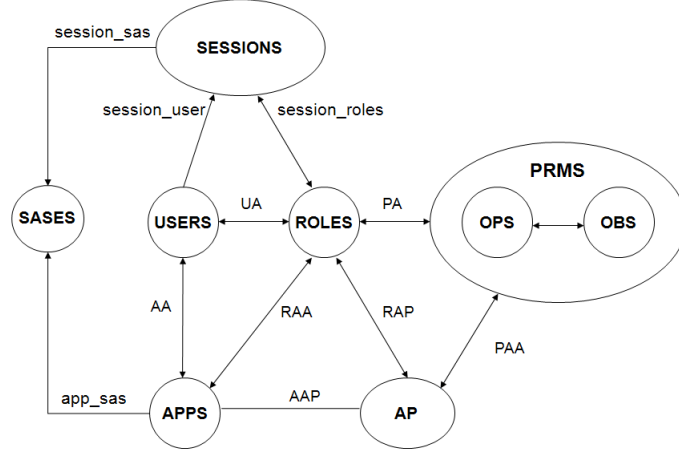


Fig. 4. Core $RBAC^+$

The application profile can also be defined as : $AP = \{n \in N, p \in N, \{next\} \in N, \{previous\} \in N\}$ with :

- $previous(p) = n$.
- $next(n) = p$.
- $previous(p) = NULL$ if p is the begin node of AP .
- $next(n) = NULL$ if n is the end node of AP .

We also define :

- $AAP : APP \rightarrow AP$, the mapping of an application onto the corresponding application profile.
-
- $RAP \subseteq ROLES \times AP$, a many-to-many mapping Role-to-Application profile Assignment relation.
- $AP_roles : AP \rightarrow ROLES$, the mapping of an application profile to a set of roles. Formally, $AP_roles(ap) = \{r \in ROLES | (r, ap) \in RAP\}$.

4.2 Sub-application session

An application session is composed of all the transactions that an application runs on behalf of all its users. A sub-application session (SASES) is the subset of transactions related to one user. Formally, we define :

Definition 3. (Sub-application session) We define:

- $app_sas : APP \rightarrow 2^{SASES}$. The mapping of an application onto a set of sub-application sessions.
- $session_sas : SESSIONS \rightarrow 2^{SASES}$. The mapping of a session onto a set of sub-application sessions.

4.3 Permissions

In our model, permissions are associated with roles and with application profiles. Applications are then associated with the appropriate roles based on the set of permissions assigned to application profiles. Such different granularities are formalized by introducing two functions: *PAA*, relating permissions and application profiles; *RAA* relating roles to applications.

Definition 4. (*Permissions*) The set of permissions *PRMS* is defined as $PRMS = 2^{(OPS \times OBJ)}$. We also define:

- $PAA \subseteq PRMS \times AP$, a many-to-many mapping *Permission-to-Application profile Assignment relation*.
- $AP_perms : AP \rightarrow 2^{PRMS}$, the mapping of an application profile onto a set of permissions. Formally, $AP_perms(ap) = \{p \in PRMS \mid (p, ap) \in PAA\}$.
- $RAA \subseteq ROLES \times APP$, a many-to-many mapping *Role-to-Application Assignment relation*.
- $APP_roles : APP \rightarrow ROLES$, the mapping of an application to a set of roles. Formally, $APP_roles(app) = \{r \in ROLES \mid (r, app) \in RAA\}$.

4.4 Users

Each user is associated with a set of applications he/she is authorized to execute. Given a set of users, the following relations are defined : the many-to-many relation *AA* relates applications and users; the function *USER_AssignedApps* maps a user onto the set of applications. More formally:

Definition 5. (*Users*) We define:

- $AA \subseteq APPS \times USERS$, a mapping *application-to-user assignment relation*.
- $USER_AssignedApps : USERS \rightarrow 2^{APPS}$, the mapping of a user to a set of applications. Formally, $USER_AssignedApps(u) = \{app \in APPS \mid (app, u) \in AA\}$.

4.5 Sessions

When a user logs in, a new session is activated and a number of roles are selected to be included in the session role set. Given a session *s*, the following three functions are defined: *session_user(s)* corresponds to the user of the session; *session_roles(s)* corresponds to the roles that are activated in *s*; *session_applications(s)* corresponds to the applications using this session. Each session is used by many applications. Given a session *s* and an application *app*, the following functions are defined: *avail_app_roles(s, app)* corresponds to the roles available for an application throughout a session; *avail_app_prms(s, app)* corresponds to the permissions available to an application in a session. Formally:

Definition 6. (*Sessions*): We define:

- *session_user*: $SESSIONS \rightarrow USERS$, the mapping from a session s to the user of s .
- *session_roles*: $SESSIONS \rightarrow 2^{ROLES}$, the mapping of session s onto a set of roles. Formally: $session_roles(s) \subseteq \{r \in ROLES \mid (session_User(s), r) \in UA\}$.
- *session_applications*: $SESSIONS \rightarrow 2^{APPS}$, the mapping of session s onto a set of applications.
- *avail_app_roles*: $(SESSIONS, APPS) \rightarrow 2^{ROLES}$, the mapping of a session and an application onto a set of roles. Formally, $avail_app_roles(s, app) \subseteq \{r \in ROLES \mid r = session_roles(s) \cap app_roles(app)\}$
- *avail_app_prms*: $(SESSIONS, APPS) \rightarrow 2^{PRMS}$, the permissions available to an application in a session. Formally, $avail_app_prms(s, app) = \bigcup_{r \in avail_app_roles(s, app)} assigned_permissions(r)$.

4.6 Access control mechanism

The session roles are the roles that are assigned to the user of the session. Therefore, the application gains access to the DBMS with the set of roles that the underlying DB user is authorized for. However, a user session contains many sessions belonging to the same application or belonging to different applications. In the first case, the roles assigned to the user are the same of those assigned to the application. In the second case, the roles to be assigned to the application represent a subset of the roles assigned to the underlying user. So, an activation of only a subset of session roles in the current application session is necessary. In order to compute the roles to be enabled, the PAA function takes as input the set of roles assigned to the user and the set of permissions composing the application profile of the underlying application and tries to find an optimum set of roles to activate so that the set satisfies the application profile permissions need and all role constraints within the system. Ideally, the chosen set of roles should activate only the requested permissions. However, this is not always possible. More formally, we define the Role Mapping (RM) problem as taking the following inputs:

- R_{avail} is the set of roles that the DB user is authorized for. This set includes roles that a user is assigned to and the roles the user can activate because of inheritance.
- P_{req} is the set of all permissions composing the application profile of an application using that DB user to access the DB.
- *perms*: gives the set of permissions each role in R_{avail} has.
- *Constraints*: \mathcal{C} represents the set of constraints that limit which roles in \mathcal{R} can be activated together. In this work, we consider the Dynamic Separation of Duty (DSoD) constraint.

A DSoD constraint $C = \langle r_1, \dots, r_n, t \rangle$, where $2 \leq t \leq n$, specifies that no single user can activate t or more roles in r_1, \dots, r_n in the same session.

4.7 Algorithm

The problem is to determine the set of roles to be activated in a single session for a particular set of permissions requested by the user. This set of roles must satisfy constraints that prevent certain combinations of roles to be activated in one session, and should follow the least privilege principle.

When determining which set of roles should be activated in a session, there are several competing concerns. First, one has to ensure that the desired permissions are covered by the roles and hence are available to the session. Second, one has to ensure that the set of roles satisfy all constraints governing role activations, such as those forbidding certain combinations of roles to be activated together in one session. Third, while satisfying the first two conditions, one wants to minimize the set of permissions that are available to the session, so as to better achieve the least privilege principle [16].

Algorithm 1, inspired from [16], implements the role mapping. The requested permissions will be exactly matched with a set of available roles. In order to improve the performance of the search, the roles with extra permissions than the requested permissions are eliminated in the beginning of search. The algorithm finds the best solution among all possible solutions. The set of selected roles which satisfy the requested permissions is denoted as R_{sel} . We define P_{rem} as $P_{req} \setminus P(R_{sel})$ which defines the permissions that have not been covered by the selected roles.

Two cases arise :

- *Case 1* : R_B matches exactly P_{req}
- *Case 2* : R_B matches P_{req} with extra permissions. In this case, the system tries to select a set of roles that cover the desired permissions while attempting to minimize extra permissions.

Definition 7. (*Authorization control function*): An access request ar is a tuple $ar = \langle U, is, app, p, o \rangle \in USERS \times ASES \times APPS \times OPS \times OBJ$. ar can be satisfied if $(p, o) \in avail_app_prms(s, a)$ and $is \in session_sas(s)$.

An sql query is a set of permissions. the above function is repeated as many permissions as the sql query requires permissions to be executed.

Definition 8. (*Path Control Function*): Let q_i be an SQL query submitted to the DBMS and q_{i-1} the SQL query submitted just before q_i . q_i can be satisfied iff $next(q_{i-1}) = q_i$.

For any SQL statement submitted, access is granted only if both authorization control function and path control function are satisfied.

5 Side Effects on other RBAC Components

As will be presented in detail in this section, $RBAC^+$ does not affect the specifications of the other three ANSI RBAC components. The main modification

Algorithm 1: RoleMatching($P_{req}, R_{avail}, C_{all}$)

Input : R_{avail} - set of roles; P_{req} - requested permissions; C_{all} - all constraints

Output: R_B - set of roles which can satisfy the exact match requirement

```
1  $R_B \leftarrow \phi$ ;  
2  $R_{sel} \leftarrow \phi$ ;  
3  $P_{rem} \leftarrow P_{req}$ ;  
4 forall  $r \in R_{avail}$  do  
5   if ( $P(r) \not\subseteq P_{rem}$ ) then  
6     remove  $r$  from  $R_{avail}$   
7   endif  
8 end  
9 selectRoles( $P_{rem}, R_{sel}, R_{avail}$ );
```

```
10 selectRoles( $P_{rem}, R_{sel}, R_{avail}$ ) { if  $P_{rem} = \phi$  then  
11   if  $R_B = \phi$  then  
12      $R_B = R_{sel}$   
13   endif  
14   ;  
15   else if  $|R_B| > |R_{sel}|$  then  
16      $R_B = R_{sel}$   
17   endif  
18   return  
19 endif  
20 forall  $C_i \in C_{all}$  do  
21   if  $|R_{sel} \cap R(C_i)|$  then  
22      $R_{avail} = R_{avail} \setminus (R(C_i) \setminus (R_{sel} \cap R(C_i)))$   
23   endif  
24   if ( $P(r) \cap P_{lb} = \phi$ ) then  
25     remove  $r$  from  $R_{avail}$   
26   endif  
27 end  
28 if  $R_{avail} = \phi$  then  
29   return  
30 endif  
31 for next  $r$  in  $R_{avail}$  do  
32   if  $P(R) \cap P_{rem} = \phi$  then  
33      $R_{avail} = R_{avail} \setminus r$   
34   endif  
35 endfor  
36 selectRoles ( $P_{rem} \setminus P(r), R_{sel} \cup r, R_{avail} \setminus r$ );  
37 selectRoles ( $P_{rem}, R_{sel}, R_{avail} \setminus r$ );  
38 }
```

of $RBAC^+$ compared to ANSI RBAC is the concepts of application, application profile and sub-application session. Thanks to this extension, it is possible, in a RBAC session, to add traceability and correlation between web users, application users and database users, while still leaving the major relations and structures of ANSI RBAC untouched.

5.1 Hierarchical $RBAC^+$

There has been much discussion about inheritance in ANSI RBAC. In [?], it was criticized that the standard was not clear about the role-inheritance semantics. The arguments say, that a role hierarchy of $r_1 \succeq r_2$ would allow three possible interpretations:

- *User Inheritance* (UI): All the users of r_1 are also authorized for r_2 .
- *Permission Inheritance* (PI): All the permissions assigned to r_2 are also authorized for r_1 .
- *Activation Inheritance* (AI): When r_1 is activated in a session, r_2 is also automatically activated in that session. It is not possible however, to access the permissions implied by r_2 without activating r_1 .

In [15], the authors of the standard responded to the raised critique, stating that the ANSI RBAC standard treats user and permission inheritance as integrated concepts. This means that role inheritance in ANSI RBAC always involves both user and permission inheritance. According to the standard authors, activation inheritance as introduced by the researchers from Purdue does not exist in ANSI RBAC, as it is possible to activate r_2 independently from r_1 in an inheritance relation of $r_1 \succeq r_2$.

The integration of user and permission inheritance in ANSI RBAC means that the inheritance relation is not affected by $RBAC^+$. Inheritance is considered as metadata to help in role mapping.

5.2 Static Separation of Duty (SSD)

$RBAC^+$ only affects roles to be enabled for an application in a session. Static Separation of Duty, on the other hand, has to be performed by the system security administrator at design time. Under SSD a user will not even be assigned to conflicting roles, which consequently means that the enabling for only a subset of roles of each application within a session does not affect this concept.

5.3 Dynamic Separation of Duty (DSD)

In contrast to Static Separation of Duty (SSD), Dynamic Separation of Duty (DSD) occurs on the session level at activation time. The RBAC ANSI standard [14] defines DSD as collection of pairs $(rs, n) = dsd$ in the set $DSD \subseteq (2^{ROLES} \times N)$, where each rs is a role set and n a natural number ≥ 2 , with the property that no user may activate n or more roles from the given set rs for each $dsd \in DSD$. As DSD affects the set of roles activated by the session user, so it affects only the role mapping function and does not affect $RBAC^+$.

6 Conclusion and Future Work

In this paper we have presented *RBAC*⁺, an extension of the RBAC model addressing access control requirements for RBAC-administered web databases. We do not only monitor DB users to detect potential attacks, but timely stop the attacks when they are detected to minimize losses caused by the attacks. However, although *RBAC*⁺ decreases the attack surface of web database by filtering numerous attacks from the access control stage, there are still attacks that cannot be prevented. Hence, we propose, as a future work, a second security layer to provide intra query validation and inter query validation represented respectively by the SQL content of an analyzed session and the values of user parameters passed in SQL sentences. Each feature supports different type of information that can be used to classify an analyzed session as legitimate or intrusion. Thus, we can build a very accurate and effective intrusion detection system that will be able to detect attacks that were undetected by other existing intrusion detection systems.

This structure represents the most effective security practice called “defense-in-depth”, which says that in order to supply information system security we must employ multiple layers of protection. The purpose of the first layer is to prevent intrusions and the purpose of the second is to detect. As future work, we will focus also on the implementation of a prototype of the proposed system and its experimentation against a variety of simulated intrusions to prove its effectiveness and efficiency.

References

1. Spalka, A., Lehnhardt, J.: A comprehensive approach to anomaly detection in relational databases. In: DBSec, pp. 207-221 (2005)
2. Valeur, F., Mutz, D., Vigna, G.: A learning-based approach to the detection of SQL attacks. In: Julisch, K., Krügel, C. (eds.) DIMVA 2005. LNCS, vol. 3548, pp. 123–140. Springer Verlag, Heidelberg (2005)
3. Lee, S.Y., Low, W.L., Wong, P.Y. Learning fingerprints for a database intrusion detection system. In: ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security London. pp. 264-280, Springer-Heidelberg (2002)
4. Hu, Y., Panda, B.: Identification of malicious transactions in database systems. In: Proceedings of the International Database Engineering and Applications Symposium (IDEAS) (2003)
5. Bertino, E., Kamra, A., Terzi, E.: Intrusion detection in rbac administered databases. In: Proceedings of the Applied Computer Security Applications Conference (ACSAC) (2005)
6. Chung, C., Gertz, M., Levitt, K.: Demids: a misuse detection system for database systems. In: Integrity and Internal Control in Information Systems: Strategic Views on the Need for Control. IFIP TC11 WG11.5 Third Working Conference (2000)
7. Lee, V., Stankovic, J., Son, S.: Intrusion detection in real-time databases via time signatures. In: Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS) (2000)

8. Chung, C.Y., Gertz, M., Levitt, K.: Misuse detection in database systems through user profiling. In a Web Proceedings of the 2nd International Workshop on the Recent Advances in Intrusion Detection(RAID). (1999)
9. Y. Zhong, X. Qin, Research on Algorithm of User Query Frequent Itemsets Mining, Proceedings of the Machine Learning and Cybernetics, pp 1671–1676 (2004)
10. D. Barbara, R. Goel, S. Jajodia : Mining Malicious Data Corruption with Hidden Markov Models, IFIP WG 11.3 Working Conference on Data and Application Security, pages 175-189 (2002).
11. Shu Wenhui, Tan T H, Daniel null, "A Novel Intrusion Detection System Model for Securing Web-based Database Systems," compsoc, pp.249, 25th Annual International Computer Software and Applications Conference (COMPSAC'01), 2001.
12. Dafydd Stuttard and Marcus. The web application hacker's handbook: discovering and exploiting security flaws, John Wiley & Sons, Inc. 2007.
13. A Classification of SQL Injection Attacks and Countermeasures
14. American National Standards Institute, Inc.: American National Standard for Information Technology - Role Based Access Control (ANSI INCITS 359-2004), 2004.
15. Ferraiolo, D., Kuhn, R., Sandhu, R.: RBAC Standard Rationale - Comments on "A Critique of the ANSI Standard on Role-Based Access Control". IEEE Security and Privacy, 51-53, November/December 2007.
16. G. T. Wickramaarachchi, Wahbeh H. Qardaji, Ninghui Li. An Efficient Framework for User Authorization Queries in RBAC Systems. Proceedings of the 14th ACM symposium on Access control models and technologies SACMAT09, June 35, 2009, Stresa, Italy. ACM.