An Access Control Model for Web Databases

Ahlem Bouchahda-Ben Tekaya, Nhan Thanh, Adel Bouhoula, Faten Labbene-Ayachi

An Access Control Model for Web Databases

Ahlem Bouchahda-Ben Tekaya[1,2], Nhan Le Thanh[1], Adel Bouhoula[2], and Faten Labbene-Ayachi[2]

[1] I3S Laboratory, Nice-Sophia Antipolis University / CNRS,
Bâtiment Polytech'Sophia - SI 930 route des colles,
B.P. 145 06903 Sophia-Antipolis France Cedex
[2] Digital security research unit,
Higher communications school of Tunis,
Rte de Raoued Km 3,5 2083, Ariana Tunisia

Abstract. The majority of today's web-based applications are based on back-end databases to process and store business information. Containing valuable business information, these systems are highly interesting to attackers and special care needs to be taken to prevent them from malicious accesses. 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, providing them by only the needed roles and continuously monitoring them throughout a whole session. It is based on business application logic rather than primitive reads and writes to enhance the ability of detecting malicious transactions. Hence, attacks caused by malicious transactions can be detected and canceled timely before they succeed.

1 Introduction

Nowadays, Web applications depend more and more on the back-end database to provide much more functionalities. Containing valuable business information, these systems are highly interesting to attackers and special care needs to be taken to prevent any malicious access to this database layer. 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. 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. It implies that the database does not identify the real user who accesses it and so, 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.
To protect web databases from attacks, access control policies should be based on a strong model that is implemented by the DBMS. Checking for authorization should be done on every attempt to access secure information. For that end, we extend the NIST RBAC (Role-Based Access Control) standard [1]. 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 by enforcing access control. The $RBAC^+$ monitors transactions issued by users and malicious transactions are viewed as intrusion behaviors. If a malicious transaction is identified, the $RBAC^+$ cancels the transaction before it succeeds, thus minimize damage caused by malicious transactions.
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 an overview of our model. We define formally and detail our model in Section 4. Access control policies are presented in section 5. We conclude our work and present future work in Section 6.

2 Related Work

The problem of access control to databases accessible over the web is very important one. This problem is well known to the web application developers and security consultants, but little existing work has addressed it. Gertz et al. in [2] pose this problem and presented some fundamental concepts and techniques that help administrators and security personnel to gradually evaluate and improve the security of a database. Also, Roichman in [3] proposed a method that uses the databases' built-in access control mechanisms enhanced with Parameterized Views and adapts them to work with web applications in order to prevent intrusions. He defines also the concept of session vector to represent the application fingerprints used in an application session in order to detect intrusions. This concept is similar to our application profile concept with the difference that the session vector represents the session profile while the application profile represents an execution way of the application. Beyond these two approaches, 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 [4]. The second consists on using Intrusion Detection Systems (IDSs). [5] presents a database IDS that uses the profile of the transactions implemented by database applications (authorized transactions) to identify user attempts to execute unauthorized transactions. Although we believe that database IDSs 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 access 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. 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.

3 Overview of our approach

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. 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. A sub-application session allows to the DBMS distinguishing between web users working with the database. It will also allow distinguishing between the requests of different web users that belong to the same database session. 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.

We assume that the user is identified at each tier. The recent tendency in the architecture of web applications allows preserving the identity of the real user

through the middle tier As example, Oracle9i introduced n-tier authentication [6], i.e., that is "lightweight session". 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. 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. 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.

4 The $RBAC^+$ Core Model

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. The general structure of the model is illustrated in Figure 1. 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.

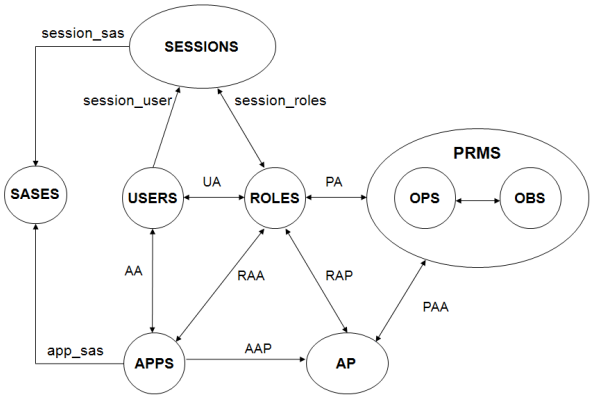

Fig. 1. Core $RBAC^+$

4.1 Application Profile

An application has many possible ways of execution each of which is called an application profile representing one valid execution sequence of the application (sequences of selects, inserts, updates, deletes). It consists of a sequence of nodes such that from each of its nodes there is an edge to the next node in the sequence. Each node in the path represent a SQL statement. It has one start node and one end node where the application execution starts and finishes, respectively. The other nodes in the path are called internal nodes. An application profile may include cycles representing the repetitive execution of sets of commands. Obviously, the set of application profiles must cover the different database application functionalities. To that end, we choose to analyze the application code which implicitly contains a policy that allows for distinguishing legitimate and malicious queries. Also, the source code contains enough information to infer models of the expected, legitimate sequences of SQL queries generated by the application.

Definition 1. *(Application Profile). An application profile AP is a binary vector with the length equal to the number of permissions (PRMS) in the DBMS, where the i^{th} bit is 1 if the application profile needs the permission p_i , else bit i is 0. $p_i \in PRMS$.*

We also define :

- $AAP : APP \rightarrow AP$, the mapping of an application onto its corresponding application profiles. Formally, $APP\_profiles(app) = \{ap \in AP | (ap, app) \in AAP\}$.
- $RAP \subseteq ROLES \times AP$, 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, $AP\_roles(ap) = \{r \in ROLES | (r, ap) \in RAP\}$.
- $RAA \subseteq ROLES \times APP$, 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, $APP\_roles(app) = \{r \in ROLES | (r, app) \in RAA\}$.

Note that, $AP\_roles(ap) \subseteq APP\_roles(app)$ with $\{ap \in AP | (ap, app) \in AAP\}$

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 2. *(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.

Definition 3. *(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,* $AP\_perms(ap) = \{p \in PRMS | (p, ap) \in PAA\}$.

4.4 Users

Each user is associated with a set of applications he/she is authorized to execute.

Definition 4. *(Users) We define:*

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

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. Formally:

Definition 5. *(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:* $session\_roles(s) \subseteq \{r \in ROLES | (session\_User(s), r) \in UA\}$.
- $session\_applications : SESSIONS \rightarrow 2^{APPS}$, *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,* $avail\_app\_roles(s, app) \subseteq \{r \in ROLES | r = session\_roles(s) \cap app\_roles(app)\}$
- $avail\_app\_prms : (SESSIONS, APPS) \rightarrow 2^{PRMS}$, *the permissions available to an application in a session. Formally,* $avail\_app\_prms(s, app) = \bigcup_{r \in avail\_app\_roles(s,app)} assigned\_permissions(r)$.

5 Access control policies

Application profiles representing valid application execution paths are used to detect unauthorized SQL statements, which are seen as invalid sequences of SQL commands.

Definition 6. *(Authorization control function): An access request ar is a tuple* $ar = \langle U, is, app, p, o\rangle \in USERS \times SASES \times APPS \times OPS \times OBJ$. 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.

To support a differentiated and adequate protection of the information stored in a database made available over the web, the access control policies must be flexible enough to support a spectrum of web-based applications, such as e-commerce web applications or health care web-applications. To that end, we specify two access control policies. The first consists on monitoring all the SQL statements submitted by an user and the second consists on monitoring only special statements that we call "critical points". A critical point may be an SQL statement manipulating sensitive data or an SQL statement modifying data (insert, delete, update). When a malicious transaction is detected, the transaction is rolled back.

5.1 Policy 1

Under this policy, every command executed must match a profile. When the first command of the transaction is executed the tool searches for all the application profiles starting with that same command, which are marked as candidate profiles for the current transaction. The next command executed is then compared with the second command of these candidate profiles. If it corresponds, the access is granted, either, access is denied. Only those who match remain candidate profiles. This process is executed over and over until the transaction reaches its end or there are no more candidate profiles for that transaction. In this latter case the transaction is identified as malicious. In practice, to detect malicious transactions, we implement the following generic algorithm over the application profiles:

```
1: while (1) do
2:   for each new command submitted do
3:     if user does not have any active transaction then {command is the 1st
       command in a new transaction } then
4:       obtain list of authorized application profiles starting with this com-
         mand
5:     else
6:       for each valid (authorized) trans. for the user do
7:         if the current command represents a valid successor node in the
           application profile then
8:           the command is valid
```

```
 9:          else
10:            mark the current transaction as a non valid trans.
11:          end if
12:        end for
13:        if there are transactions marked as non valid then
14:          a malicious transaction has been detected
15:        end if
16:      end if
17:    end for
18:  end while
```

5.2 Policy 2

Under this policy, the SQL statements submitted by the user called *user context* are stored and the access is granted until he submits a critical point. In this case, the tool searches an application profile that corresponds to the user context. If one or more corresponding application profiles are found, the access is granted. Otherwise, the transaction is rolled back.

6 Conclusion and Future Work

Database security problems seriously persecute web-based applications that rely on web databases storing invaluable data. 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. As future work, we will focus 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. : American national standard for information technology, role based access control. ansi incits 359-2004 (February 2004)
2. Gertz, M., Gandhi, M.: Security Re-engineering for Databases: Concepts and Techniques. In: Handbook of Database Security. (2007) 267–296
3. Roichman, A.: Intrusion prevention and detection for web databases (2008)
4. Halfond, W.G., Viegas, J., Orso, A.: A classification of sql-injection attacks and countermeasures. In: Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA (2006)
5. Vieira, M., Madeira, H.: Detection of malicious transactions in dbms. In: PRDC '05: Proceedings of the 11th Pacific Rim International Symposium on Dependable Computing, Washington, DC, USA, IEEE Computer Society (2005) 350–357
6. Oracle corporation: Oracle9i database concepts release 2 (9.2). chapter 22: Controlling database access. Available at: http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96524/c23acces.htm