

Enhanced Role based Access Control: Integrating Auditing and Authentication

Kriti

M.Tech, CSE

Manav Rachna International University, Faridabad,
Haryana, India

Indu Kashyap

Assistant Professor, CSE

Manav Rachna International University, Faridabad,
Haryana, India

ABSTRACT

In past decade lot of research has been done in RBAC (Role Based Access Control) technology. The industries have also shown great interest in RBAC. Most of the IT vendors are offering products that incorporate some form of RBAC. Today, all major DBMS products support RBAC. RBAC provides easier management of permissions in an organization and hence is most widely used model to control access of legitimate users. However research shows that access control is not a complete solution for securing a database. Most of the breaches are done by insiders. So, access control system must be incorporated with other mechanisms that provide more features than just controlling access of users. Auditing is such a mechanism that can log all the transactions occurring on the database and based on this log an analysis can be done. Auditing is well effective when we have good authentication. Authentication processes are vulnerable to SQL Injection attacks. This paper proposes an enhanced model that increases the capability of RBAC model by integrating Auditing and Authentication in simplest ways. In this way this model not only provides the features of RBAC but also handles common issues of database security.

Keywords

RBAC, Auditing, SQL Injection, DBMS_FGA, Authentication, Prepared Statements.

1. INTRODUCTION

RBAC is a proven technology for large-scale authorization. However lack of a standard model results in uncertainty and confusion about its utility and meaning [1]. Because of its relevance in products and applications for the management of enterprise security, RBAC always has been the focus of standardization activities. The American National Standard Institute (ANSI) standard was approved in 2004 and provided a consistent and uniform definition of RBAC [2]. This ANSI RBAC standard consists of two parts: a Reference Model and a Functional Specification. The Reference Model defines sets of basic RBAC elements and relations, and the Functional Specification specifies the operations and functions an RBAC system should support. The RBAC model and functional specification are organized into four RBAC components: Core RBAC, Hierarchical RBAC, Static Separation of Duty (SSD) Relations, and Dynamic Separation of Duty (DSD) Relations (Constrained RBAC).

The goal of RBAC systems is to provide a model and tools to help manage access control in complex environment with a very large number of users and even larger number of data

items [6]. A recent study by National Institute of Standards and Technology (NIST) demonstrates that RBAC addresses many needs of the commercial and government sectors [3] [7]. Most of the organizations base their access control decisions on the roles that individual users take on as part of the organization. Other evidence of strong interest in RBAC comes from the standards arena. Roles are being considered as part of the emerging technologies. RBAC is also well matched to prevailing business trends. A number of products support some form of RBAC directly, and others support closely related concepts, such as user groups, that can be utilized to implement roles.

However, access control is also not a complete solution for securing a system; it must be coupled with auditing [4].

Auditing helps in following ways:

- It performs the analysis of all the requests and activities of the users in the system.
- It performs logging of all users request and activities for their later analysis.
- Auditing ensures that authorized user do not misuse their privileges.
- Auditing can determine flaws in the security system.

In other words auditing holds users accountable for their actions. Also effective auditing requires good authentication to be in place.

Access control is different from authentication. Authentication is a process of signing on to a computer system by providing an identifier and a password. So, correctly establishing the identity of the user is the responsibility of the authentication service. On the other hand, access control assumes that authentication of the user has been successfully verified prior to enforcement of access control. Authentication process is vulnerable to SQL Injection attacks mostly in case of web applications. In these attacks attacker tries to add malicious code to legitimate SQL query and try to gain access to resources or database.

So a model is needed that not only provides features of RBAC but also should be invulnerable to SQL Injection attacks. Also it should provide some mechanism to keep an eye on the activities that an individual user perform in system. In this paper a model - Authentication & Auditing enabled Role

Based Access Control (AARBAC) is presented that integrates features of RBAC with auditing and authentication mechanisms invulnerable to SQL Injection attacks. The rest of the paper is organized as follows: In section 2 standard RBAC model is discussed. In section 3 model AARBAC is discussed along with description of its various components. Sections 4 and 5 discusses about how authentication and auditing can be integrated in simplest ways with RBAC respectively. Section 6 concludes the paper.

2. OVERVIEW OF ANSI RBAC

Role-Based Access Control (RBAC) is an emerging paradigm for controlling access to computer resources. RBAC adds the notion of roles between users and permissions. Roles are created based on job functions in an organization. Permissions are assigned to roles. Users are made members of roles based on their job responsibilities and qualifications, thereby gaining permissions assigned to those roles. In RBAC, users are granted permissions based on their roles, not on individual basis. This abstraction provided by roles significantly simplifies the management of privileges, and helps enforcing the principle of least privilege [2] [8].

2.1 Terms & Definitions

- **Component:** It refers to one of the major blocks of RBAC features, core RBAC, hierarchical RBAC, SSD relations, and DSD relations [5].
- **Objects:** An *object* can be any system resource subject to access control, such as a file, printer, terminal, database record, etc.
- **Operations:** An *operation* is an executable image of a program, which upon invocation executes some function for the user.
- **Permissions:** *Permission* is an approval to perform an operation on one or more RBAC protected objects.
- **Role:** A *role* is a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role.
- **User:** A *user* is defined as a human being. Although the concept of a user can be extended to include machines, networks, or intelligent autonomous agents, the definition is limited to a person in this document for simplicity reasons.

2.2 RBAC Reference Model

The RBAC reference model is defined in terms of four *model components*—Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations [5].

Core RBAC defines a minimum collection of RBAC elements, element sets, and relations in order to completely achieve a Role-Based Access Control system. This includes user-role assignment and permission-role assignment relations, considered fundamental in any RBAC system. In addition, Core RBAC introduces the concept of role activation as part of a user's session within a computer system. Core RBAC is required in any RBAC system, but the other components are independent of each other and may be implemented separately.

The Hierarchical RBAC component adds relations for supporting role hierarchies. A hierarchy is mathematically a

partial order defining a seniority relation between roles, whereby senior roles acquire the permissions of their juniors and junior roles acquire users of their seniors. In addition, Hierarchical RBAC goes beyond simple user and permission role assignment by introducing the concept of a role's set of authorized users and authorized permissions.

A third model component, Static Separation of Duty Relations, adds exclusivity relations among roles with respect to user assignments. Because of the potential for inconsistencies with respect to static separation of duty relations and inheritance relations of a role hierarchy, the SSD relations model component defines relations in both the presence and absence of role hierarchies.

The fourth model component, Dynamic Separation of Duty Relations, defines exclusivity relations with respect to roles that are activated as part of a user's session.

2.3 Core RBAC

Core RBAC includes sets of five basic data elements called users (USERS), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS). The RBAC model as a whole is fundamentally defined in terms of individual users being assigned to roles and permissions being assigned to roles. As such, a role is a means for naming many-to-many relationships among individual users and permissions. In addition, the core RBAC model includes a set of sessions (SESSIONS) where each session is a mapping between a user and an activated subset of roles that are assigned to the user [4].

Core RBAC model element sets and relations are defined in Figure 2.1 which also illustrates *user assignment (UA)* and *permission assignment (PA)* relations. The arrows indicate a many-to-many relationship (i.e., a user can be assigned to one or more roles, and a role can be assigned to one or more users). This arrangement provides great flexibility and granularity of assignment of permissions to roles and users to roles.

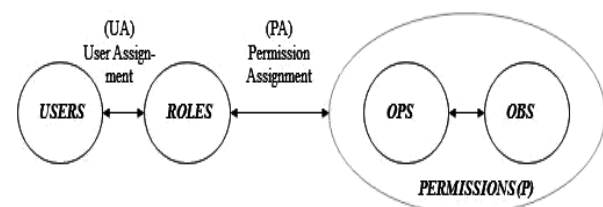


Figure 2.1: Core RBAC

2.4 Hierarchical RBAC

This model component introduces role hierarchies (RH) as indicated in Figure 2.2. Role hierarchies are commonly included as a key aspect of RBAC models and are often included as part of RBAC product offerings. Hierarchies are a natural means of structuring roles to reflect an organization's lines of authority and responsibility [5].

Role hierarchies define an inheritance relation among roles. Inheritance has been described in terms of permissions; i.e., $r1$ "inherits" role $r2$ if all privileges of $r2$ are also privileges of $r1$.

ANSI RBAC recognizes two types of role hierarchies—general role hierarchies and limited role hierarchies. General role hierarchies provide support for an arbitrary partial order to serve as the role hierarchy, to include the concept of multiple inheritances of permissions and user membership among roles. Limited role hierarchies impose restrictions resulting in a simpler tree structure (i.e., a role may have one

or more immediate ascendants, but is restricted to a single immediate descendent).

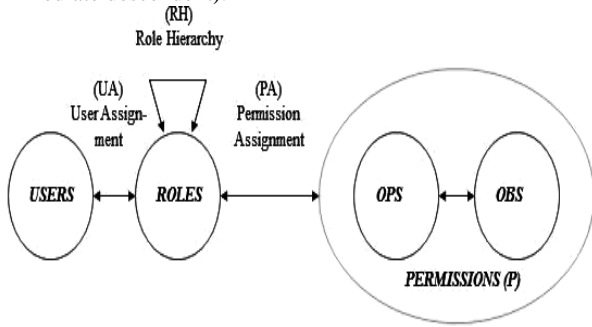


Figure 2.2: Hierarchical RBAC

2.5 Constrained RBAC

Constrained RBAC adds Separation of Duty relations to the RBAC model. Separation of duty relations are used to enforce conflict of interest policies that organizations may employ to prevent users from exceeding a reasonable level of authority for their positions. ANSI RBAC allows for both static and dynamic separation of duty as defined within the next two subsections.

2.5.1 Static Separation of Duty Relations

Conflict of interest in a role-based system may arise as a result of a user gaining authorization for permissions associated with conflicting roles. One means of preventing this form of conflict of interest is through *static separation of duty*, that is, to enforce constraints on the assignment of users to roles.

The static constraints defined in this model are limited to those relations that place restrictions on sets of roles and in particular on their ability to form UA relations. This means that if a user is assigned to one role, the user is prohibited from being a member of a second role. An SSD policy can be centrally specified and then uniformly imposed on specific roles. From policy perspective, static constraint relations provides a powerful means of enforcing conflict of interest and other separation rules over sets of RBAC elements. Static constraints generally place restrictions on administrative operations that have the potential to undermine higher-level organizational Separation of Duty policies [5].

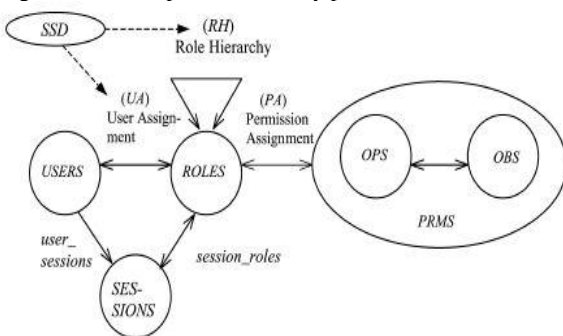


Figure 2.3: Static Separation of Duty Relations

2.5.2 Dynamic Separation of Duty Relations

DSD relations differ from SSD relations by the context in which these limitations are imposed. This model component defines DSD properties that limit the availability of the permissions over a user’s permission space by placing constraints on the roles that can be activated within or across a user’s sessions. DSD properties provide extended support for the principle of least privilege in that each user has different levels of permission at different times, depending on the role being performed. These properties ensure that permissions do

not persist beyond the time that they are required for performance of duty. This aspect of least privilege is often referred to as *timely revocation of trust*. Dynamic revocation of permissions can be a rather complex issue without the facilities of dynamic separation of duty, and as such it has been generally ignored in the past for reasons of expediency [5].

This model component provides the capability to enforce an organization-specific policy of dynamic separation of duty (DSD). DSD allows a user to be authorized for two or more roles that do not create a conflict of interest when acted in independently, but produce policy concerns when activated simultaneously. For example, a user may be authorized for both the roles of Cashier and Cashier Supervisor, where the supervisor is allowed to acknowledge corrections to a Cashier’s open cash drawer. If the individual acting in the role Cashier attempted to switch to the role Cashier Supervisor, RBAC would require the user to drop the Cashier role, and thereby force the closure of the cash drawer before assuming the role Cashier Supervisor. As long as the same user is not allowed to assume both of these roles at the same time, a conflict of interest situation will not arise. Dynamic separation of duty relations are defined as a constraint on the roles that are activated in a user’s session (Figure 2.4).

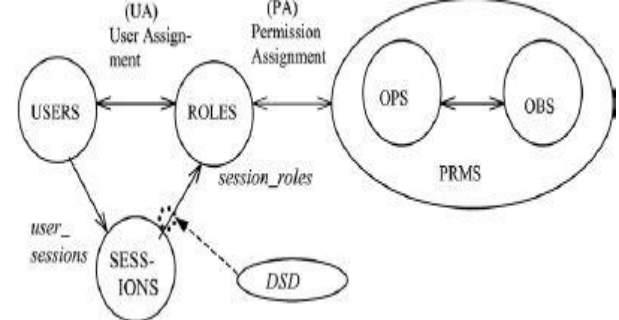


Figure 2.4: Dynamic Separation of Duty Relations

3. ENHANCED MODEL: AARBAC

Access control is also not a complete solution for securing a system; it must be coupled with auditing. Also effective auditing requires good authentication to be in place. In this section a model is presented that integrates RBAC with Authentication and Auditing module. This model is called AARBAC (Authentication & Auditing Enabled Role Based Access Control). This model not only provides feature of RBAC but also handles and SQL injection attacks effectively. For simplicity Hierarchical RBAC is integrated with these modules. However it is possible to integrate any form of RBAC.

3.1 Components of AARBAC

- **Administrator:** A **database administrator** or **DBA** is a person responsible for the installation, configuration (configuration is an arrangement of functional units according to their nature and characteristics), upgrade, administration, monitoring (is a process of collecting and analyzing data) and maintenance of databases in an organization.
- **Authorization Database:** This database contains authorization information i.e. the information about all the data items and authorizations the users of system have on those items. This database is consulted to find out if a user is attempting to do an operation is actually authorized to do that operation or not.

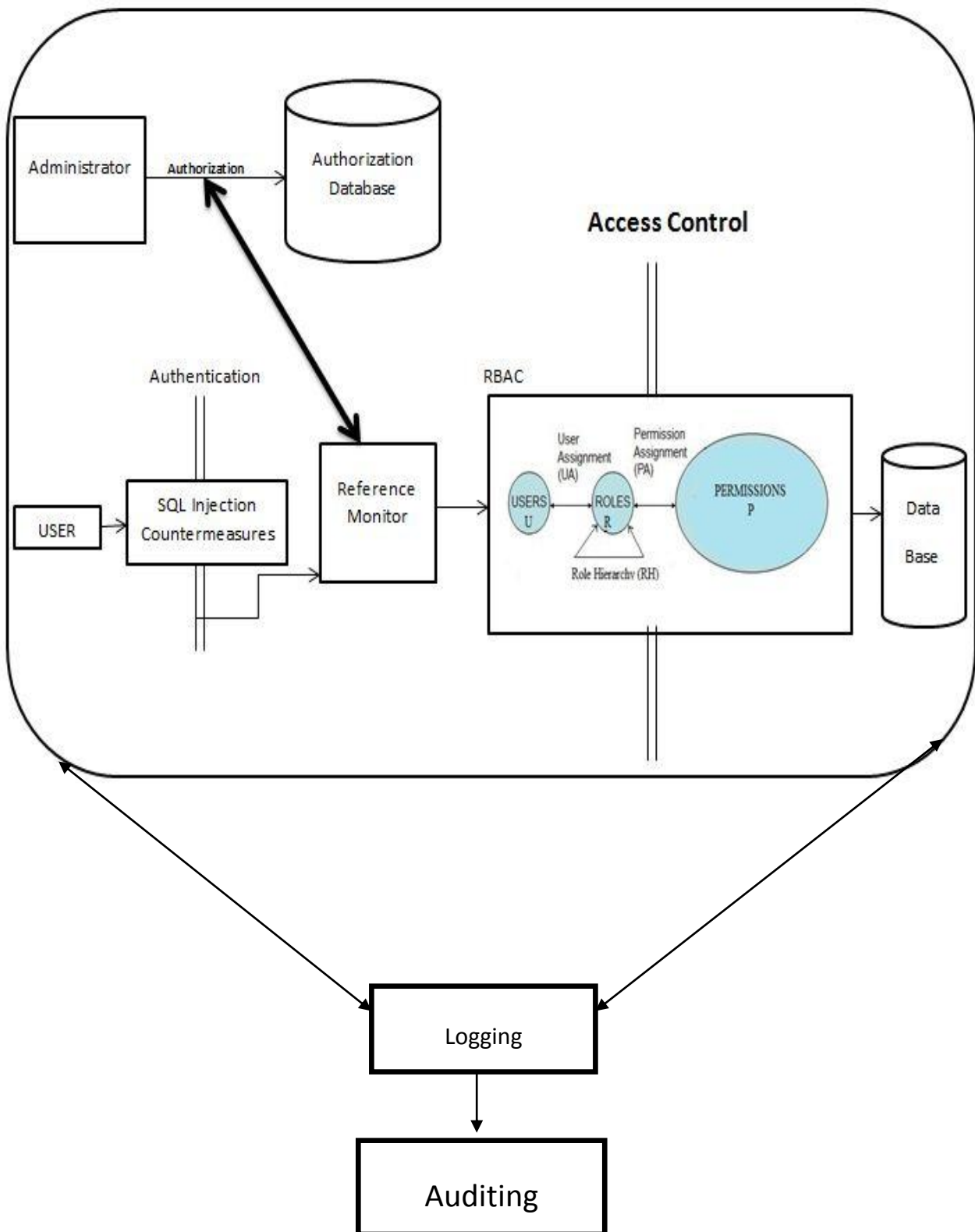


Figure 3.1: Enhanced Model- AARBAC (Authentication & Auditing enabled Role Based Access Control)

- **User:** Any person or a process executing on the behalf of user who wants to gain access to database items.
- **Authentication:** It is a process of signing on to a computer system by providing an identifier and a password. So, correctly establishing the identity of the user is the responsibility of the authentication service.
- **SQL Injection Countermeasures:** Mechanism that prevents SQL injection attacks during authentication procedure.
- **Reference Monitor:** A program executing on the behalf of users which mediates every attempted access, consults authorization database etc.

- **Access Control:** The process of limiting access to the system only to authorized users, programs etc.
- **RBAC:** Role-Based Access Control (RBAC) is an emerging paradigm for controlling access to computer resources. RBAC adds the notion of roles between users and permissions. Roles are created based on job functions in an organization. Permissions are assigned to roles. Users are made members of roles based on their job responsibilities and qualifications, thereby gaining permissions assigned to those roles. In RBAC, users are granted permissions based on their roles, not on individual basis. This abstraction provided by roles significantly simplifies the management of privileges, and helps enforcing the principle of least privilege.
- **Role:** A job function within the organization that describes the authority and responsibility conferred on a user assigned to the role.
- **Permission:** A description of the type of authorized interaction a subject can have with an object.
- **Role Hierarchy:** An inheritance relationship established among roles.
- **Database:** A passive entity that contains objects and information e.g. Files, tables, resources etc.
- **Auditing:** Auditing is the recording and analyzing of events to provide information about system use and performance in a clear and understandable manner. The goal of an auditing system is to be able to determine if security and other policies are being violated.
- **Logging:** It is a part of auditing. The logger or logging procedure records information for analyzing e.g. login date and time, user's role etc.

3.2 Advantages of AARBAC

- Provides features of RBAC, which is the most widely used access control model in industries.
- Authentication is done prior to authorization.
- Authentication is free from SQL Injection attacks.
- An auditing mechanism is there which keeps track of all transactions occurring on database.

4. AUTHENTICATION IN AARBAC

4.1 SQL Injection Attacks

In SQL Injection attacks the attacker attempts to modify the existing SQL statement by adding elements to the WHERE clause [9] [10]. Most of the SQL Injection attacks occur during the login authentication.

A simplistic web application may check user authentication by executing the following query:

```
SELECT * FROM users
WHERE username = 'bob' and PASSWORD = 'mypassword'
```

The attacker attempts to manipulate the SQL statement to execute as:

```
SELECT * FROM users
WHERE username = 'xxx' or 'b' = 'b' and PASSWORD =
'xxxx' or 'a' = 'a'
```

Based on operator precedence, the WHERE clause is true for every row and the attacker has gained access to the application [11].

4.2 Preventing SQL Injection Attacks: Use of Prepared Statements

The Prepared Statement is a more powerful version of a Statement. Most relational databases handle a JDBC / SQL query in four steps:

- Parse the incoming SQL query.
- Compile the SQL query.
- Plan/optimize the data acquisition path.
- Execute the optimized query / acquire and return data.

A Statement will always proceed through the four steps above for each SQL query sent to the database. A Prepared Statement pre-executes steps (1) - (3) in the execution process above. Thus, when creating a Prepared Statement some prior optimization is performed immediately. The effect is to lessen the load on the database engine at execution time.

One more benefit of prepared statement is to prevent SQL Injection by making them impossible in database through the use of bind variables [13]. An example is shown below [12] [14]:

```
PreparedStatement pstmt =conn.prepareStatement ("SELECT
* FROM users WHERE username = (?) and PASSWORD =
(?)");
pstmt.setString (1, usern);
pstmt.setString (2, PASSW);
pstmt.execute();
```

Here “?” is the bind variable that binds username to variable usern and PASSWORD to PASSW.

4.3 Example

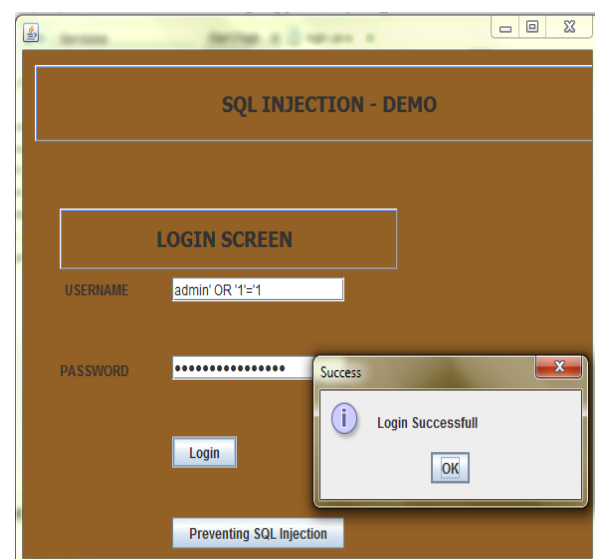
Consider an example of a simple login procedure by providing credentials. When the prepared statements are not used system is vulnerable to SQL Injection and database is breached as shown below [15]:

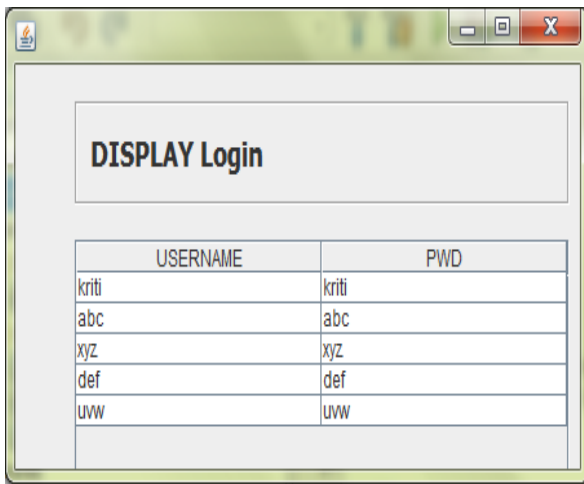
```
stmt = conn.createStatement();
```

```
String sql = "select * from login where username='"+un+"'
AND pwd='"+pwd+"'";
```

```
rst = stmt.executeQuery(sql);
```

Here prepared statements are not used so result is:





USERNAME	PWD
kriti	kriti
abc	abc
xyz	xyz
def	def
uvw	uvw

Figure 4.1: SQL injection Attack

Now prepared statements are used to prevent SQL Injection Attack as follows:

```
String query = "Select * from login where username = ? AND
                pwd = ?";

                if (pstFlag)
                {
                    pstmt = conn.prepareStatement(query);

                    pstmt.setString(1, un);

                    pstmt.setString(2, pwd);

                    rst = pstmt.executeQuery();
                }
```

Here flag is a Boolean variable which is initially set false, when the user clicks on “Preventing SQL Injection” this flag becomes true and the control enters this loop that uses prepared statements preventing SQL Injection. Now the result is (Figure 4.2).

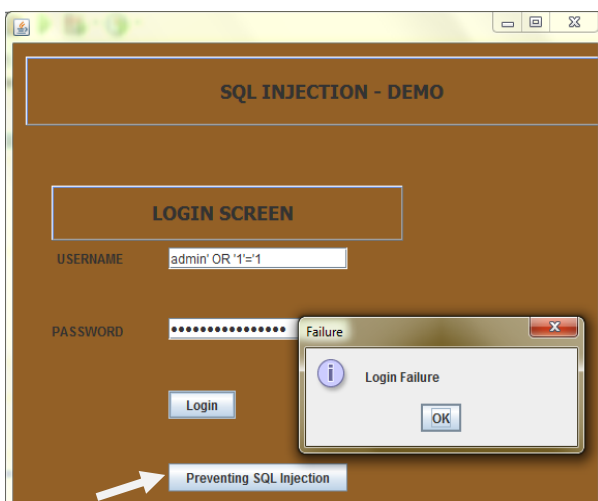


Figure 4.2: SQL Injection Attack Prevented

5. AUDITING AND LOGGING IN AARBAC

Auditing is the recording and analyzing of events to provide information about system use and performance in a clear and understandable manner. The goal of an auditing system is to

be able to determine if security policies are being violated. Given an audit log of “who” did “what” to “which” data “when” and “how” and an effective means of processing the log, auditing can answer “why”. By answering the question of “why”, it provides one of the means to detect intrusions into the system [16].

Malicious access to protected data can cause major damage to organizations and also problems for the public. According to study on insider threat by US Secret Service and CERT CC in 2005, 29% of the incidents of intrusion of database system are done by individuals inside organizations that have privileges to freely access data. Therefore, auditing systems must include the ability to prevent and detect internal threats to the system [17].

5.1 Basic architecture of auditing system

The basic architecture of an auditing system is [16]:

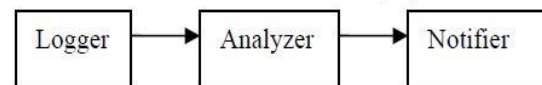


Figure 5.1 Components of Auditing System

- The logger records information. The information that is to be stored in the log is determined by the security policies.
- The analyzer takes the log as the input and analyzes it to either determine if an event of interest or a problem has occurred, or if other information needs to be logged.
- The notifier receives the analysis from the analyzer, and reports the result of the audit.

5.2 Logger

Some important issues related to logging include what, when, where, how, and how often a database should be logged. The ways in which the databases address these issues have an impact on other aspects of the system.

When and how often to log: Audit module should monitor every command to detect qualified events. If a particular event is met with the predefined condition by users, it logs necessary information to the database. This is the fundamental principle of the database audit. It is also necessary to keep an eye on events every time to avoid breach of information. Periodic auditing is also useful depending upon the needs of an organization.

What to log: It is better to log more and more information about an event but this is not always feasible. It is good to keep track of username, name of the schema object accessed, date and time stamp etc.

Where to log: Oracle permits the administrator to choose the location of audit trail in either an operating system file or a database. Oracle uses an audit trail database named SYS.AUD\$ and DATA DICTIONARY table. The SYS.AUD\$ database has several tables such as DBA_AUDIT_SESSION, DBA_AUDIT_OBJECT, and DBA_AUDIT_STATEMENT. To make data retrieval easy, several built-in views are provided to users. DBA_AUDIT_TRAIL is a view based on the AUD\$ table that presents the information in a readable, text based form. Users are also able to log to the operating system’s file. That is done by use of syslog function. This option is more secure since the Database Administrator (DBA) does not have privilege to delete the OS’s syslog. This means there is still evidence of malicious users’ activities even if the DBA

modified some tables or records on purpose and then deleted audit information from the native database.

How to log: One very important auditing feature of Oracle is Fine-Grained Auditing (FGA). Normal auditing triggers can detect who modified which table or which column. In other words, users can track any data modification on tables if they write a trigger procedure. Unfortunately, this does not hold for “SELECT” statement since “SELECT” does not change any data, no trigger can be made. If the security administrator wants to monitor the person who accesses a particular table, more specifically a column which contains sensitive data, he can use Fine-Grained Auditing feature (FGA). Whenever specified columns are accessed, Oracle logs that event into FGA audit trail. Then an analyst can check the table whether a security breach has occurred or not. Let’s see now how a FGA policy can be written.

5.3 DBMS-FGA

DBMS_FGA is needed for administering audit policies. Because the audit function can potentially capture all user environment and application context values, policy administration should be executable by privileged users only [18] [19].

ADD_POLICY Procedure Creates an audit policy using the supplied predicate as the audit condition. This procedure creates an audit policy using the supplied predicate as the audit condition.

Syntax

```
DBMS_FGA.ADD_POLICY (
object_schema VARCHAR2,
object_name VARCHAR2,
policy_name VARCHAR2,
audit_condition VARCHAR2,
audit_column VARCHAR2,
handler_schema VARCHAR2,
handler_module VARCHAR2,
enable BOOLEAN,
statement_types VARCHAR2,
audit_trail BINARY_INTEGER IN DEFAULT,
audit_column_opts BINARY_INTEGER IN DEFAULT);
```

Example

```
begin
dbms_fga.add_policy (
object_schema=>'BANK',
object_name=>'ACCOUNTS',
policy_name=>'ACCOUNTS_ACCESS',
audit_column => 'BALANCE',
audit_condition => 'BALANCE >= 11000'
);
end;
```

In this example, there is a database ‘BANK’ which contains a table named ‘ACCOUNT’. That table has a column ‘BALANCE’. If administrator wants to monitor when a user tries to access records whose balance is more than \$11000, Oracle logs this event to the FGA audit trail.

5.4 Analyzer & Notifier

The ability to analyze the data in the audit logs contributes to the effectiveness of the auditing system. Since Oracle uses tables as a location of logs, users can easily access and investigate them using standard SQL command. Generally the analysis work is done by administrator or the person with highest privileges.

After analysis it is the duty of analyzer to notify the result. If the analysis represents any flaw in DBMS it should be notified to all the users of DBMS by the analyser else if the analysis shows any breach the analyser should take appropriate actions.

6. CONCLUSION

Role Based Access control models are widely in organizations for access control. In RBAC access control decisions are often based on the roles individual users take on as part of an organization. A role specifies a set of transactions that a user or set of users can perform within the context of an organization. RBAC provide a means of naming and describing relationships between individuals and rights, providing a method of meeting the secure processing needs of many commercial and civilian government organizations. Various forms of role based access control have been described under ANSI standardization. Core RBAC defines features that are minimally required of all RBAC systems. Hierarchical RBAC add requirements of role hierarchy. Hierarchies are a natural means of structuring roles to reflect an organization’s lines of authority and responsibility. Constrained RBAC adds Separation of Duty relations to the RBAC model. Separation of duty relations are used to enforce conflict of interest policies that organizations may employ to prevent users from exceeding a reasonable level of authority for their positions. RBAC simplifies the access control but access control is considered to be only a partial solution to provide security. This paper integrates auditing and authentication modules to RBAC model and enhanced its capability to provide security. This model is called AARBAC. Authentication is different from access control. Authentication is a process of signing on to a computer system by providing an identifier and a password. So, correctly establishing the identity of the user is the responsibility of the authentication service. Most of the authentication procedures are affected by SQL Injection attacks. SQL Injection attacks are code modifying attacks. Prepared statements can be used to prevent these attacks. These statements use bind variables that make SQL Injection attacks impossible. Auditing is a process of finding whether a security policy is being violated or not by logging and analyzing events. Oracle provides feature of fine grained auditing using DBMS_FGA policy. These two modules can be integrated with any system incorporating RBAC mechanism to enhance its functionality.

7. REFERENCES

- [1] Sandhu, Ravi, David Ferraiolo, and Richard Kuhn, The NIST model for role-based access control: Towards a unified standard. Symposium on Access Control Models and Technologies, Proceedings of the fifth ACM workshop on Role-based access control, Volume 26, No. 28, 2000, pp. 47- 63.
- [2] Li N., J. Byun, and E. Bertino, A critique of the ANSI standard on role-based access control, Security & Privacy, IEEE Volume. 5, no. 6, 2007, pp. 41-49.
- [3] R. Sandhu and E. J. Coyne, Role-Based Access Control Models, IEEE Computer, 1996, pp. 38-47.
- [4] R. Sandhu and P. Samarati, Access Control: Principles and Practice, IEEE, Communications Magazine, 1994, pp. 40-48.
- [5] ANSI, American national standard for information technology – role based access control. ANSI INCITS 359-2004, February 2004.

- [6] S. L. Osborn, Role- Based Access Control, Springer, Security, Privacy, and Trust in Modern Data Management, 2007, pp. 55-70.
- [7] R. Sandhu, E. J. Coyne H. L. Feinstein and C. E. Youman, Role – Based Access Control Models, IEEE Computer, Volume 29, No. 2, February 1996, pp. 38-47.
- [8] D. F. Ferraiolo and R. Kuhn, Role – Based Access Control, 15th National Computer Security Conference, Baltimore, 1992, pp. 554-563.
- [9] S. Tripathi and P. Zehnde, Surveys on Vulnerabilities, Threats and Security Methods of DBMS, 3rd Biennial National Conference on Nascent Technologies, 2012, pp. 39-44.
- [10] San-Tsai Sun, Ting Han Wei, Stephen Liu, and Sheung Lau, Classification of SQL Injection Attacks, University of British Columbia, 2007, pp. 1-6.
- [11] Halfond, W. G., Jeremy Viegas, and Alessandro Orso, A classification of SQL-injection attacks and countermeasures, In Proceedings of the IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, 2006, pp. 1-11.
- [12] S. Kost, An Introduction to SQL Injection Attacks for Oracle Developers, Integrity Corporation, March 2007, pp. 5-25.
- [13] S. Thomas , L. Williams and Tao Xie, On Automated Prepared Statement to remove SQL Injection Vulnerabilities, Elsevier, Information & Software Technology, No. 51, 2009, pp. 589-598.
- [14] R.P. Mahapatra and S. Khan, A Survey Of Sql Injection Countermeasures, International Journal of Computer Science & Engineering Survey (IJCSES), Volume 3, No.3, June 2012, pp. 55-74.
- [15] Z. Raveshi, S. R. Idate, Investigation and Analysis of SQL Injection Attacks on Web Applications: Survey, International Journal of Engineering and Advanced Technology (IJEAT), Volume-2, Issue-3 February 2013, pp. 182-187.
- [16] J. Woo, S. Lee, C. Zoltowiski, Database Auditing.
- [17] DB Audit for Oracle, Microsoft SQL Server, Sybase ASE, Sybase ASA, and IBM DB2, SoftTree Technologies.
- [18] T. Bednar, Oracle Database Auditing: Performance Guidelines, An Oracle White Paper, August 2010.
- [19] Oracle Database, PL/SQL Packages and Types Reference, 10g Release 2 (10.2), B14258-02, pp. 40(1) – 40(13).