Efficient Isolation Enabled Role-Based Access Control for Database Systems

Mohammad Rahat Helal, Weiqing Sun, Ahmad Y Javaid

College Of Engineering
The University of Toledo, Toledo, OH-43606, USA
{mohammadrahat.helal, weiqing.sun, ahmad.javaid}@utoledo.edu

Abstract—In this paper, we propose designing and implementing Isolation enabled Role-Based Access Control (RBAC) at the record level in a database. The concept involves integrating isolation concepts of a relational database management system (RDBMS) into the NIST RBAC model in a secure and efficient way. A user is denied access in a typical RBAC system if the system recognizes the user as an unauthorized user. Our proposed system allows the user limited access to the system instead of complete denial in the case of an emergency or an unavoidable circumstance. One such example being, the senior role could delegate restricted access to the junior role. Using this restricted access, the junior role can perform actions which are mandatory to be performed on behalf of the senior role. The system has been designed in a way to keep it secure, efficient, available and consistent. The proposed system enhances security of and ease of access to the system in absence of an authorized user by restricting unauthorized user access to only an isolated view of the database. Moreover, in that scope of access, he/she can perform actions that are isolated from other users. The paper presents design and implementation of the concept, and compares our work with the approach followed by other RBAC implementations.

Keywords—Host System, Isolated System, Transaction Isolation levels, Access Control Systems

I. INTRODUCTION

Considering the current landscape across all organizational domains including industry, governance, businesses and academia, it has become a priority to ensure security of a system along with its availability and efficiency. Role Based Access Control [1] offers the flexibility of adding users to an organization, and associates the users with their respective roles, while permissions are assigned on objects based on their roles. Such a system offers a two-fold advantage over other access control mechanisms in terms of ease of user management and data/network security. Access to a system should not be compromised at any time. RBAC offers robust security to the system in a way that it denies access to unauthorized user and grants access to authorized users only.

Roles and permissions on objects are set up according to the rules, policies and business logic of any organization. Therefore, it could vary from organization to organization and sometimes within an organization as well. However, we encounter certain situations when a particular user, say a senior role in the organization, is absent and an emergency demands him/her to access the system. Although an employee in a junior role might be available to perform the operation, lack of access would prohibit the junior employee from updating/changing the system. There could also be real time scenarios where inter departmental access of data is required in a limited way, such as in a healthcare facility. In an IT service industry scenario, the junior developer roles can code and build in the development environment, with limited or no permission to migrate the code from development to other coding environments. These situations could lead to delays in required work to be performed, resulting in losses to business or much needed information not being available to the end user. Another example, in a healthcare system, access control model to be set up should not deny access to a user in case of doubt, since the best possible care being provided to the patients is the primary goal there [2]. Therefore, a patient's electronic record should be available for access across departments, with limited and appropriate access indeed, to tackle emergencies.

In conclusion, such issues during emergencies, if not resolved quickly, could result in problems ranging from loss of human life to delay in the delivery of a product.

We propose to design and implement Isolation enabled Role-Based Access Control (I-RBAC) at the record level in a database. The idea could be applied to the way data is being stored in an organization, its retrieval, and manipulation and reporting. The main goal of this paper is to integrate isolation concepts in a database transaction into the RBAC model, and thereby, adding usability and efficiency to it.

The rest of the paper is organized as follows. Section II provides an overview and design of our system. Section III describes the implementation and evaluation of the system. Related Work is discussed in Section IV. Limitations of the proposed system have been discussed in Section V. Finally, we conclude in Section VI.

II. SYSTEM DESIGN

The system that we are proposing would use RBAC layer, i.e. Host system, and an isolated layer over it, which is a copy of Host system, for allowing access to the unauthorized users [3]. The Host system could be defined as the production environment of an organization which is accessed by the users and customers, to read and write information as required based on the access rights granted to them. The Isolated system is like the Host system in terms of design and concept. Some of the differences being - addition of attributes to the objects (database tables in our case) that differentiates both the systems, access policy checks/rules, i.e. authorization rules that differentiates the access to Host and Isolated system, control dependency of refreshing the Host system as and when required, etc.
We have taken a health-care facility as an example for our implementation, where a doctor is designated as a senior role, who has all the required access to all objects in the Host System. An intern doctor is a junior role, who has no or limited access to any object in the Host System [4]. Therefore, an intern doctor is granted access to the Isolated System to perform required operations in the absence of the doctor. The records created/manipulated by him/her have to go through security checks in the Isolated System itself before they are reviewed by the doctor. Once reviewed by the doctor, the Host System is refreshed with the new/modified records in the Isolated System.

While the Isolated System offers an alternate approach to access the system, preserving the security of data is our primary goal. An intern is not able to manipulate Host System records, he/she would be automatically redirected to the Isolated System once he/she logs into the system, as shown in Fig. 1. For the intern, roles and permissions on objects are limited to the Isolated System only. We have added certain attributes to the custom tables for the patient record, prescription file, medication report etc. in the database. These attributes are used to track different operations in the isolated environment. The attributes are: “Record Create Date”, “Record Last Update Date”, “Record Updated By User”, “Record Version Number” and “Active Indicator”. They are created alongside other attributes of custom tables, and are populated according to the logic of addition/modification of records in the Isolated System (shown in Fig. 1). Therefore, using them we can track who/when created/modified a particular record in the Isolated System. In addition, if a patient record is undergoing changes multiple times, the highest version number of the record and a non-zero active indicator would tell us the most recent record. This also gives us a history of events associated with a particular record, e.g. editing certain fields to update by more than one intern. The security check is performed after addition/modification of records in the isolated system, which makes sure that properly accumulated modifications have been done in the system.

While the Isolated System offers an alternate approach to access the system, preserving the security of data is our primary goal. An intern is not able to manipulate Host System records, he/she would be automatically redirected to the Isolated System once he/she logs into the system, as shown in Fig. 1. For the intern, roles and permissions on objects are limited to the Isolated System only. We have added certain attributes to the custom tables for the patient record, prescription file, medication report etc. in the database. These attributes are used to track different operations in the isolated environment. The attributes are: “Record Create Date”, “Record Last Update Date”, “Record Updated By User”, “Record Version Number” and “Active Indicator”. They are created alongside other attributes of custom tables, and are populated according to the logic of addition/modification of records in the Isolated System (shown in Fig. 1). Therefore, using them we can track who/when created/modified a particular record in the Isolated System. In addition, if a patient record is undergoing changes multiple times, the highest version number of the record and a non-zero active indicator would tell us the most recent record. This also gives us a history of events associated with a particular record, e.g. editing certain fields to update by more than one intern. The security check is performed after addition/modification of records in the isolated system, which makes sure that properly accumulated modifications have been done in the system.

Fig. 1: System Design Flow Chart

To capture database transactions capability in Isolation, when multiple users/sessions are trying to access the same table, we created two sessions and performed different types of transaction isolation in a relational database. For example, taking the default transaction isolation of “Repeatable Read” in MySQL, we observed that the second session can view the record of the first session only after the commit operation is performed by the first session, i.e. these two sessions would not be able to see the other’s changes. In our implementation perspective, if two interns are trying to access the same record, it would be “copy on write” for them and future accesses would be redirected to this isolated copy for a consistent view. A copy of the record from Host System would be created in the Isolated System if it does not exist already, and this would be available for modifications by the intern. If there is more than one record in the Isolated System for a given patient, then the most recent record would be fetched for the intern to work upon.

A. Design Steps

1) Design a basic level RBAC model which allows users (medical personnel) to view, edit, add and delete records in the system as per the permission assigned to their roles on different objects.
   - Create tables for USERS, ROLES, PERMISSIONS and OBJECTS.
   - Using the USERS and ROLES tables, create USER ASSIGNMENT table. This table assigns roles to users. When a new user needs to be added into the system, he/she needs to be added into the USERS table. The USER ASSIGNMENT could be updated with the role assigned to the user.
   - Using the ROLE, PERMISSION and OBJECT tables to create PERMISSION ASSIGNMENT table. The level of access granted to a role is defined here. For example: An intern has Read/View access to an electronic patient record and no access to patient’s insurance information.
   - Create RBAC Policy table to define authorization of users through Role, Permission and Object combination. For example: An intern is authorized to Read/View electronic patient record, but he/she is not authorized to delete the record for a patient.
   - Create custom tables for the electronic patient record and prescription file to realize the basic implementation of the Isolation concepts.

2) Design Isolated system layer in conjunction with the RBAC layer that allows restricted access as per policy described in the Isolated system.
   - Create a table for I-RBAC login, define user credentials having access to the Isolated System/environment.
   - Create I-RBAC Policy table to define authorization of users through Role, Permission and Object combination in the Isolated System. For example: An intern is authorized to view, edit, create an electronic patient record, but he/she is not authorized to delete the record for a patient in the Isolated System.
   - Create the Isolated version/copy of the Host custom tables - electronic patient record and
prescription file, to copy the records to and from Host environment before/after modifications.

3) Define attributes and scenarios in the Isolated system which would differentiate the two environments and would be used to insert, update and retrieve records as per different scenarios.

- While creating/replicating the custom tables, add the attributes - Load Date, Last Updated Date, Record Version Number, Active Indicator and Updated By (User) to the tables. These attributes would differentiate both the environments. This has been explained in detail in the System Implementation section.
- These would be used to maintain and retrieve records from the isolated system. For example: two interns can edit the same prescription file record and these two records would be stored in the Isolated System. Fig. 2 shows two users working on the same record in the Isolated System, followed by security checks performed by Host user before the records are merged to Host System. The Isolated users can retrieve the records for further set of operations and each time they access the record, they would be redirected to the isolated copy of the record created by them.

4) Refresh the Host System with the modifications performed in the Isolated System after getting them approved by the authorized personnel/user (depicted in Fig. 2).

- The system would perform post policy checks and security checks on the modified records and validate them at database level, based on the rules defined for the records. For example: An intern can delete only the prescription file which was created by him/her.
- The modifications performed in the Isolated System by the intern would be merged to the Host System after verification and validations by the senior employee.

III. SYSTEM IMPLEMENTATION

The implementation of the overall system consists of creating the Host and Isolated system for RBAC and I-RBAC system. For realization of RBAC and I-RBAC system implementation, we propose to create both the systems under one database only, i.e. different tables of the two systems would be created in the same database, and the Isolated System tables would be prefixed with Isolation System parameter. So, if PRESCRIPTION_FILE table holds information of patient’s prescription files in Host System, then I_PRESRIPTION_FILE would contain the data of the same patient in Isolated System. We have used Programming Language PHP for server side scripting (HTML used for markup and client side scripting). Database used is MariaDB (an open source variant of MySQL).

A. Database level Implementation

As mentioned in the System Design section, tables created for realization of RBAC system are shown in Fig. 3. Tables created in the Host System has been shown on the left side, in rectangle, while tables specifically for Isolated System are shown in oval outline. The different tables used to support Users, Roles, Permissions, Objects, User Assignment, Permission Assignment and other system tables are described as follows:

- USER_LOGIN - Holds login information of all users (username and password).
- USER_RBAC - Extended version of Login table which holds information such as Last Name, First Name, and Address, etc.
- ROLE_RBAC - Consists of Roles and Role Description, currently we have taken Doctor and Intern Doctor as two roles.
- PERMISSION_RBAC - Consists of Permissions and its description, currently we have taken View, Edit, Create and Delete as permissions.
- OBJECTS - Holds information of different objects available to the user, such as Prescription file and Patient Record.
- USER_ASSIGNMENT - Holds information of mapping of Users and their respective Roles.
- PERMISSION_ASSIGNMENT - Holds information of Roles and related permissions on the objects assigned to the roles.
- RBAC_POLICY - This table would define rules for users to be authorized/unauthorized to access the RBAC system and access to the OBJECTS.
• **IRBAC_LOGIN** - The login information present for users who are defined as isolated users, intern doctors in our case.

• **IRBAC_POLICY** - The rules for users in the isolated system defined in this table.

• **PRESCRIPTION_FILE** - An organization specific table containing information for patients prescription files in our case.

• **I_PRESCRIPTION_FILE** - The version of patient’s prescription file contained in the Isolated System.

As mentioned in the System Design section, in addition to the fields in the Host environment table (for example: PRESCRIPTION_FILE), a few new fields would be added when the same table is copied/cloned in the Isolated System.

- **RCRD_LOAD_DATE** - A date field. Whenever a record is inserted into the Isolated System, this field would take the current date and time for that record in the isolated system.

- **RCRD_LAST_UPDATED** - An auto generated date-time field. Whenever any user updates a record in the Isolated System, this field would get updated for that record.

- **RCRD_VRSN_NUM** - A SMALLINT field. As the user performs any update on existing record, a version number would be assigned to it, which would be assigned in an incremental fashion. Initially all records copied from Host table would have a value of 1, but after a particular record gets updated, the version number for that record increases by 1.

- **ACTV_IND** - A TINYINT field. The purpose is to indicate only one record as active at a given time. As the record is updated it would have Active Indicator ACTV_IND = 1, and the previous version of the same record would have ACTV_IND = 0.

- **RCRD_UPDT_BY** - The identification of the user who works on the particular record, this would be used while retrieving the record for each user.

Once the unauthorized user logs into the system, he/she is redirected to the Isolated System. In our case we have taken “Intern Doctor” as unauthorized role. The webpage displays the User’s Role, his/her authorized (access to Host system) and unauthorized (access to Isolated System) access/permissions on the objects and records on the right side of the screen depicting the records created/modified in the Isolated System. Fig. 4 displays results for one such intern doctor. For this user, there are no authorized roles and permissions, and he/she can perform his/her operations in the Isolated System.

**B. Logic of data flow**

The logic for the operations to be performed and the considerations for an object such as a Prescription file, in the Isolated System could be summarized as below:

- **The Host environment would have only one record for a patient that is the source of truth for both the environments.**

- **The Isolated environment can have more than one record; it depends on the number of times it has been modified by the unauthorized users.**

The conditions for viewing the prescription file for any operation to be performed by the intern in the Isolated System:

- **Condition 1**: The file is in Isolated, but not in Host, which means it was created by the Intern Doctor (unauthorized role) and now it needs to be approved and pushed to Host by the Doctor.

- **Condition 2**: The file is in Host but not in Isolated, which means it needs to be copied to the Isolated system from...
Host and the Intern would perform required operations on it.

- **Condition 3**: Only one copy of the file is present in both the environments, ideally they both should be the same. When the user accesses the file, he/she should be directed to the isolated record and he/she is required to make any changes as per requirements.

- **Condition 4**: One copy of the file is in Host and multiple copies of that file are in Isolated, i.e. the file has undergone changes several times by different users at different times. Therefore, the most recent and active record in the Isolated System would be shown to the intern for any operations to be performed.

**Algorithm:** Above conditions for a particular record:

Variables defined:

\[
\begin{align*}
\text{Count}_{\text{Host}} &= \text{CountOf(record\_in\_Host)} \\
\text{Count}_{\text{Isolated}} &= \text{CountOf(record\_in\_Isolated)}
\end{align*}
\]

begin
\[
\begin{align*}
\text{if Count}_{\text{Host}} > 0 \text{ then} \\
\quad \text{if Count}_{\text{Isolated}} == 0 \text{ then} \\
\qquad \text{INSERT RECORD FROM HOST TO ISOLATED} \\
\quad \text{if Count}_{\text{Isolated}} == 1 \text{ then} \\
\qquad \text{SELECT RECORD FROM ISOLATED} \\
\quad \text{if Count}_{\text{Isolated}} > 1 \text{ then} \\
\qquad \text{SELECT ALL RECORDS FROM ISOLATED ORDER BY CREATED\_DT DESC OR APPROPRIATE ISOLATED CONDITIONS} \\
\text{else if Count}_{\text{Host}} == 0 \text{ then} \\
\quad \text{if Count}_{\text{Isolated}} == 0 \text{ then} \\
\quad \quad \text{Print "No records for the patient available in the system"} \\
\quad \text{if Count}_{\text{Isolated}} > 1 \text{ then} \\
\quad \quad \text{INSERT RECORD FROM ISOLATED TO HOST AFTER SECURITY CHECK AND APPROVAL}
\end{align*}
\]

Note: The above mentioned conditions for retrieving records from Isolated System are applied while copying the record from Host to Isolated and updating the records in Isolated System as well.

When a user copies a record from Host or retrieves a record in Isolated System to update, initially the Isolated System specific attributes for the record are as shown in Table I. In the Isolated System, when the record is retrieved for any update, the attribute values for the old and new record are as shown in Table II. At any given time, the parameters in Table III would show the latest record to be accessed.

### C. Realization of Isolation at database level

We have used the default transaction isolation “Repeatable Read” in MySQL for easy and efficient implementation. As mentioned in MySQL documentation about Repeatable Read: All consistent reads within the same transaction read the snapshot established by the first such read in that transaction. To get a fresher snapshot of queries we need to commit the current transaction after issuing new queries [5]. For realization of isolation, we took different scenarios from two session’s perspective to understand.

**Scenario 1**: When the first session inserts a record into a table, the second session is able to view that record only after commit operation is performed in the first session. The first session can see the records read at the beginning of the session. If the second session inserts a record into the table, the first session is unable to read it until the sessions perform commit.

This scenario provides isolation to both the sessions in a way that the second session is unable to view the record inserted by the first one, since it is still not committed.

**Scenario 2**: When both the sessions are inserting records into the same table at the same time, i.e. overlapping sessions. In this case, both the sessions would be able to view the records inserted by the other session only when commit is performed in both the sessions. Individual inserts could be viewed in individual session windows, but until both the sessions perform a commit, the database is not in a consistent state. Only after both the sessions commit, they would be able to see the newly inserted records.

This scenario depicts the situation when multiple interns are adding records into the same table, isolation system ensures that they are not able to view the records inserted by the other, until they all have committed and queried the system for all patients information in a table.

### TABLE I: Attributes in Isolated System

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Date</td>
<td>SYSDATE</td>
</tr>
<tr>
<td>Last Updated Date</td>
<td>SYSDATE</td>
</tr>
<tr>
<td>Record Version Number</td>
<td>1</td>
</tr>
<tr>
<td>Active Indicator</td>
<td>1</td>
</tr>
<tr>
<td>Updated By User</td>
<td>CURRENT_USER</td>
</tr>
</tbody>
</table>

### TABLE II: Old vs New attribute values for record in Isolated System

<table>
<thead>
<tr>
<th>Old record</th>
<th>New Record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Date = SYSDATE</td>
<td>Load Date = No change</td>
</tr>
<tr>
<td>Last Updated Date = SYSDATE</td>
<td>Last Updated Date = SYSDATE</td>
</tr>
<tr>
<td>Record Version Number = 1</td>
<td>Record Version Number = 2</td>
</tr>
<tr>
<td>Active Indicator = 0</td>
<td>Active Indicator = 1</td>
</tr>
<tr>
<td>Record Updated By User =</td>
<td>Record Updated By User =</td>
</tr>
<tr>
<td>CURRENT_USER</td>
<td>CURRENT_USER</td>
</tr>
</tbody>
</table>

### TABLE III: Parameters for latest record in Isolated System

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last Updated Date (new record)</td>
<td>Last Updated Date (old record)</td>
</tr>
<tr>
<td>Record Version Number (new record)</td>
<td>Record Version Number (old record)</td>
</tr>
<tr>
<td>Active Indicator</td>
<td>1</td>
</tr>
</tbody>
</table>
Scenario 3: When both the sessions perform an Update operation on the same record: In this case, if the transaction is affecting the same rows then those rows/records get locked. When the first session performs update on a record, the second session is in locked state until the first session commits that update. As soon as first session commits the transaction, the update operation in the second session gets performed.

The transactions get queued up at this point, the order of execution follows the sequence or serialization order, in order of occurrence/changes performed. Until the lock is released, the commands have to wait - type of FIFO operation. Only after both the sessions commit, one can view a consistent state of the database.

We tested remaining Isolation transactions levels available in MySQL - Read Committed, Read Uncommitted, Serializable - at the session and server levels, to completely understand their functioning and efficiency. We observed and calculated that “Repeatable Read” isolation at both session and server level is best suited to our proposed design. If many users are accessing the system simultaneously, we do not want the risk of them being blocked due to a lock caused by a transaction writing to a table. And at the same time we want database performance to be good. While we want data concurrency in the system, we want data to be consistent as well. Considering all this, we decided to go ahead with the optimized method of repeatable read in our proposed system.

IV. RELATED WORKS

We have seen different models of access control systems derived from RBAC, each of them addressing various access related issues and improving upon them. While each of the access control system implementation is based on specific scenarios and logic, their complexities vary greatly in terms of resource utilization, efficiency, availability, integrity and security of data and time management.

One of the latest implementation is ABAC [6] (Attribute based access control) where the main concept is to use labeled objects and user attributes instead of permissions to provide access in a flexible manner. This approach considers attributes such as time of the day and user location for granting access. Prior to this approach, we have witnessed separate implementations of time and location based approach as discussed here. These methods provide simplified and scenario specific implementation of RBAC and therefore, they have their own set of advantages and disadvantages.

One such implementation is TRBAC [7] (Temporal Role Based Access Control) which implements RBAC for organizations where work/shift timings of resources are considered. For example, in a typical healthcare facility, nurses and intern doctors should be able to carry out the tasks meant to be performed by the doctors on night duty or shifts. Similarly, their access should be made unauthorized beyond the shift intervals. This process demands enabling or disabling the accesses of the junior roles by some authorized personnel each time the shift changes or upon request by the senior officials.

Another model based on RBAC proposed is related in the context of location of the user. This is generally referred to as Spatial Role-based access control [8] (SRBAC). If a doctor wants to access some patient record, which he must access in his personal cabin, then the doctor must refrain from using any sort of mobile devices in public places. This requires the access model to be able to locate the user and allow required access to the system.

There is another model which is integrated into RBAC and allows the system to grant access to unauthorized users in a controlled manner. This was termed as BTG [9] (Break the glass). This model is similar to the method adopted by us. It can handle emergency situations like our proposed model, but it requires a separate RBAC system to be built for unauthorized roles, which makes the entire implementation an expensive and time consuming approach.

By comparing these access control systems, we conclude that adoption of an isolated system as suggested by us is quite simple, inexpensive, efficient and easily maintainable. The rules created for Isolated System are provided by the organization and they can be implemented without much hassle. The unauthorized users would be redirected to isolated system always, thereby, maintaining the security of the system. Also, there would not be any timing constraints related to system access, resulting in efficient access to resources at all times. The I-RBAC system can be implemented as a layer over the existing RBAC model as explained in this paper. Therefore, on one hand it is easy to implement, and on the other hand it is as secure. It takes care of system availability, efficiency and integrity at all times.

V. LIMITATIONS

As we are dealing with a database system for the implementation of our model, there are few related limitations of this approach. When the model is deployed for a large-scale organization which has many users, say in millions, then such a system gets tough to manage. Though the database is capable of handling multiple transactions at the same time, sometime it may be difficult to handle complicated scenarios. Such scenarios could be related to database server, handling locks on data during multiple simultaneous transactions, infrastructure or network related issues. We have not tested our approach for such large scale user database in any of these categories. While the Isolated System keeps on growing day after day, it would be necessary to archive or purge the historic data on regular intervals. Since, we would be interested in keeping history, the database architect should consider the options carefully while designing such a system.

Even in a simple implementation involving a few million records in the isolated system, it becomes necessary to index the tables, gather statistics, create and query views rather than underlying tables, and tune complex queries for report generation. All these tasks demand proper steps to be taken while designing the software itself. The database architect should give proper attention to detail of each of the subject tables and its integrity with other tables in the system.

VI. CONCLUSION

In this paper, we have suggested an alternate approach to understand and implement RBAC policies. The model proposed by us integrates and implements the isolation principles of
database into RBAC model at record level and allows access to unauthorized user to objects in Isolated System. The changes performed by the unauthorized user in Isolated System are posted to the Host System only after they undergo Security Policy Checks and is approved by the authorized user or a senior official in the organization. We incorporate several attributes which are specific to the Isolated System for identification of records. In addition, we identified different scenarios of performing database operations on the records in Isolated System. The transaction capability of database in isolation ensures secure and efficient handling of data in the Isolated System. The method proposed by us strives to resolve the access related issues faced in adopting other implementations of RBAC. The implementation results in robust and efficient system which provides data security in the Host as well as the Isolated System.

REFERENCES


