Trusted RUBIX™
Version 6

Application User Guide
Revision 2

RELATIONAL DATABASE MANAGEMENT SYSTEM

Infosystems Technology, Inc.

4 Professional Dr - Suite 118

Gaithersburg, MD 20879

TEL +1-202-412-0152

Infosystems Technology, Inc.
4 Professional Dr - Suite 118
Gaithersburg, MD 20879

The information in this document is subject to change without notice and should not be construed as a commitment by ITI.

Infosystems Technology, Inc. assumes no responsibility for any errors that may appear in this document.

Printed in U.S.A.

Overview of Application User Mechanism

The Trusted RUBIX (TR) Application User mechanism eliminates many vulnerabilities present in RDBMS applications, such as SQL injection, URL manipulation, and application hijacking. It accomplishes this by extending Mandatory Access Controls (MAC) to the TR application's users. With other RDBMSs', security enforcement for this class of user, including authentication and access controls, are typically performed entirely within the RDBMS application. Because RDBMS applications tend to be ad-hoc, custom programs, they do not have the well-structured, evaluated security mechanisms typical of RDBMS. Additionally, with other RDBMS, the existence of application users is unknown to the RDBMS.

The Mandatory Access Control (MAC) over the application users is accomplished by securely binding the application users' authentication to the TR application's database session and then using Attribute Based Access Control (ABAC) security policies to restrict access of individual application users down to the row level. In a typical configuration, this would result in application users only being able to access rows which they created. Additionally, the TR user executing the application program would not be able to access any row unless the application user that created the row is currently authenticated.

Using the Trusted RUBIX Application User mechanism, TR application developers can focus on the functionality of the application while relying on the Trusted RUBIX RDBMS to encapsulate and secure the underlying data of a secure transaction based web application, e.g., Internet Banking.

This document will describe the basic concepts, operation, and administration of the Trusted RUBIX Application User mechanism.

Application User Concepts

RDBMS User

A Trusted RUBIX RDBMS User represents a traditional user of the database. The RDBMS User is able to authenticate to the RDBMS, initiate a RDBMS session, and directly submit SQL operations. The Trusted RUBIX RDBMS User space is tightly integrated with the user space of the operating system that hosts the database server software (i.e., the database host platform). As such, the user name, user ID, group membership, group names, group ID's, and security label (MLS, TE, RBAC) are identical on both the operating system and the RDBMS. Additionally, the RDBMS User is authenticated using the Pluggable Authentication Module of the host platform. The Trusted RUBIX RDBMS User is controlled by SQL Discretionary Access Control (DAC), Multilevel Security (MLS), Attribute Based Access Control (ABAC), Type Enforcement (TE, SELinux only), and Role Based Access Control (RBAC). The RDBMS User is available to the ABAC mechanism as an attribute for security decisions.

Applications

A Trusted RUBIX Application represents a RDBMS middleware application. It is named and is created by the Trusted RUBIX Database Administrator. It has one or more associated Application Administrators and Application Users. Application Users, typically connecting from the Internet, represent users of the middleware application. Application Administrators are RDBMS users who are permitted to execute the RDBMS middleware application, connect to the database, and submit SQL operations on behalf of

Application Users.

APPLICATION ADMINISTRATORS

A Trusted RUBIX Application Administrator is a RDBMS user that is permitted to execute a particular RDBMS middleware application (e.g., ODBC program), connect to the database, and submit SQL operations on behalf of Application Users. The Trusted RUBIX Security Administrator assigns RDBMS users to be Application Administrators for a particular Application.

Typically, the Application Administrator executes the middleware application, connects to the database, declares itself to be administering the Application (i.e., use the ALTER SESSION SET APPLICATION command), and then services operations on behalf of Application Users. The Trusted RUBIX RDBMS will ensure that only RDBMS users that have been assigned as administrators for the particular Application may use the ALTER SESSION SET APPLICATION command. Furthermore, it will only allow Application Users associated with the Application to authenticate to the RDBMS while the Application is set as current in the session.

During the execution of the middleware application the current Application, Application Administrator, and Application User are available as security attributes for Attribute Based Access Control (ABAC) security decisions.

APPLICATION USERS

Application Users represent users of the RDBMS middleware application (i.e., a Trusted RUBIX Application). Typically, they connect to the Application using the Internet. Application Users may only interact with Trusted RUBIX through the single Application with which they are associated. They may not connect directly to the RDBMS. Application Users are associated with the Application when their account is created by the Application Administrator using the CREATE APPLICATION_USER command.

Application Users must authenticate to the Trusted RUBIX RDBMS. To accomplish this, the Application User submits its authentication information to the Application. The Application Administrator would then submit the authentication information to the RDBMS using the AUTHENTICATE APPLICATION_USER command. Once authenticated, the Application User is set as current in the database session and is available to the ABAC mechanism as an attribute for security decisions.

AUTHENTICATION TIMEOUT AND THE APPLICATION USER POOL

When an Application User is authenticated to the Trusted RUBIX RDBMS, it is set as the current Application User in the database session. The authentication will remain valid until the Application Administrator explicitly terminates the authentication (i.e., uses the AUTHENTICATE APPLICATION_USER with an empty password) or the authentication timeout is reached. The authentication timeout is configurable for each Application in the Trusted RUBIX configuration file (*rxconfig*).

When an Application User is authenticated it is added to the authenticated Application User pool. This is a cache of authenticated Application Users. It will remain in the pool during the entire time the authentication is valid and subsequently removed. While there may be many Application Users in the pool, only one may be current in the database session. Only the Application User that is current in the database session is available to the ABAC mechanism as an attribute for security decisions.

The Application Administrator may choose which Application User is current using the ALTER SESSION

SET APPLICATION_USER command. This command may only be used for Application Users currently residing in the authenticated Application User pool.

SUMMARY TABLE

The following table summarizes the main constructs of the Trusted RUBIX Application User mechanism.

Construct	Description	Security
RDBMS User	Traditional RDBMS user. Trusted RUBIX uses OS users as RDBMS users.	Created within the operating system by the OS Administrator. Same user ID, user name, group name, and group ID as OS user. Controlled by Standard SQL DAC, MLS, and ABAC. Current RDBMS user and group in the database session available as an ABAC security attribute.
Application	Represents a RDBMS middleware application (e.g., ODBC program). Has a string name.	Created by the Database Administrator. Has an associated set of Application Administrators and Application Users. Current Application in the database session available as an ABAC security attribute.
Application Administrator	A RDBMS user that has been assigned administrative duties for an Application. Executes the RDBMS middleware and submits SQL operations on behalf of Application Users.	The only RDBMS users that may execute an Application by setting it as current in the database session. The only RDBMS users that may submit SQL operations on behalf of Application Users.
Application User	User of the RDBMS middleware application. Typically connect from Internet. Has a string name unique to Application and numerical ID unique to the database.	May authenticate to the RDBMS only through the Application by the Application Administrator. Current Application User in the database session available as an ABAC security attribute.

MAC using Attribute Based Access Control (ABAC) and Security Policy Manager (SPM)

Security methods for controlling access to database objects have traditionally been partitioned into two categories: Discretionary Access Control (DAC) and Mandatory Access Control (MAC).

DAC allows the owner of an object to control which subjects and actions may operate upon the database object. The owner of the object may even give other subjects (i.e., non-owners) the ability to control access to the object. This method of access control is called discretionary because the owner of the object has the discretion to choose who may do what to the object.

MAC allows only the Security Administrator to control which subjects and actions may operate upon the

database object. The owner of the object has no special permissions. Typically, the mandatory security behavior is strategically planned as a primary feature of database development. MAC security environments typically have a higher level of trust and assurance. This method of access control is called mandatory because the security enforcement is mandatory for every subject. Trusted RUBIX provides three MAC security mechanism: Multilevel Security (MLS), Type Enforcement (TE, SELinux only), and Attribute Based Access Control (ABAC). The ABAC security enforcement is integrated with the Application User mechanism to provide Mandatory Access Controls over database objects and Application Users down to the row level.

The ABAC security enforcement of the Trusted RUBIX Security Policy Manager (SPM) is a general purpose policy rules engine that uses custom XML based security policies. ABAC policies calculate security decisions using a host of attributes (e.g., Application User ID, time, SQL operation, object name, client IP, etc.). The result of policy execution is to permit or deny an operation and to optionally execute specific actions, called Obligations. Obligations may be thought of as "secure triggers" and perform such actions as setting the value of a row field and setting the user observable SQL operation error code.

TYPICAL ABAC SECURITY POLICY

A typical ABAC security policy used in conjunction with the Application User mechanism will enforce two security objectives. The first is to restrict row access to the Application User that created it. This will eliminate SQL injection and URL manipulation attacks. The second security objective is to deny the Application Administrator access to a database row unless the creating Application User is currently authenticated to the Application. This will greatly reduce the damage of an application hijacking attack.

The first step in enforcing these rules is to securely label each row with the creating Application User. This is accomplished by creating the table with a column that will hold the Application User ID. Then, an ABAC security policy is written for the INSERT operation that contains an Obligation that sets the column with the current Application User ID. The following XML code snippet shows this Obligation:

```
<Obligations> <Obligation FulfillOn="Permit" ObligationId="set-field">
    <AttributeValue DataType="string">db.cat.sch.tab.app_userid</AttributeValue>
    <SubjectAttributeDesignator AttributeId="application-user-id" MustBePresent="true"/>
</Obligation> </Obligations>
```

Other code in the policy (not shown) would ensure that this code snippet would only be applied during a permitted INSERT operation. This Obligation will set the *db.cat.sch.tab.app_userid* integer field to the value of the current Application User ID, as designated by the *application-user-id* attribute ID. The *MustBePresent* option being set to *true* ensures that the Application User ID is available (i.e., the Application User is authenticated) at the time of the insert. Note that the INSERT operation will fail if there is no authenticated Application User. Also note that the *app_userid* field value will be set to the Application User ID even if the SQL INSERT operation specifies a different value for the field.

The second step in enforcing the security rules is to ensure that a row access operation is only permitted if the currently authenticated Application User created the row being accessed. The following XML code snippet show this logic:

```
<VariableDefinition VariableId="is-creator">
    <Apply FunctionId="equal">
        <SubjectAttributeDesignator AttributeId="application-user-id"/>
        <AttributeSelector RequestContextPath="db.cat.sch.tab.app_userid" DataType="integer"/>
    </Apply>
</VariableDefinition>
```

Other code in the policy would ensure that this code snippet would only be applied during a SELECT, UPDATE, or DELETE operation. The variable definition will evaluate to *true* only if the current Application User ID is equal to the value of the *db.cat.sch.tab.app_userid* field of the particular row being accessed. Note that it will evaluate to *false* if no Application User is currently authenticated. The evaluated variable definition will be consulted to make the policy decision for the SELECT, UPDATE, or DELETE operation, permitting the operation only if the variable definition evaluates to *true*.

Overview of Typical Architecture

The following diagram shows a typical account-based web application architecture. It consists of several web applications which exist on the Internet. These applications may execute within a web browser (e.g., Internet Explorer) or as a stand-alone Java application. The web application allows application users to access their account (e.g., bank account) using a user name and a password. The web application submits operations (e.g., read bank balance) to the RDBMS middleware application as the user interacts with the web application. From a security perspective, the web application should not be trusted to act properly.

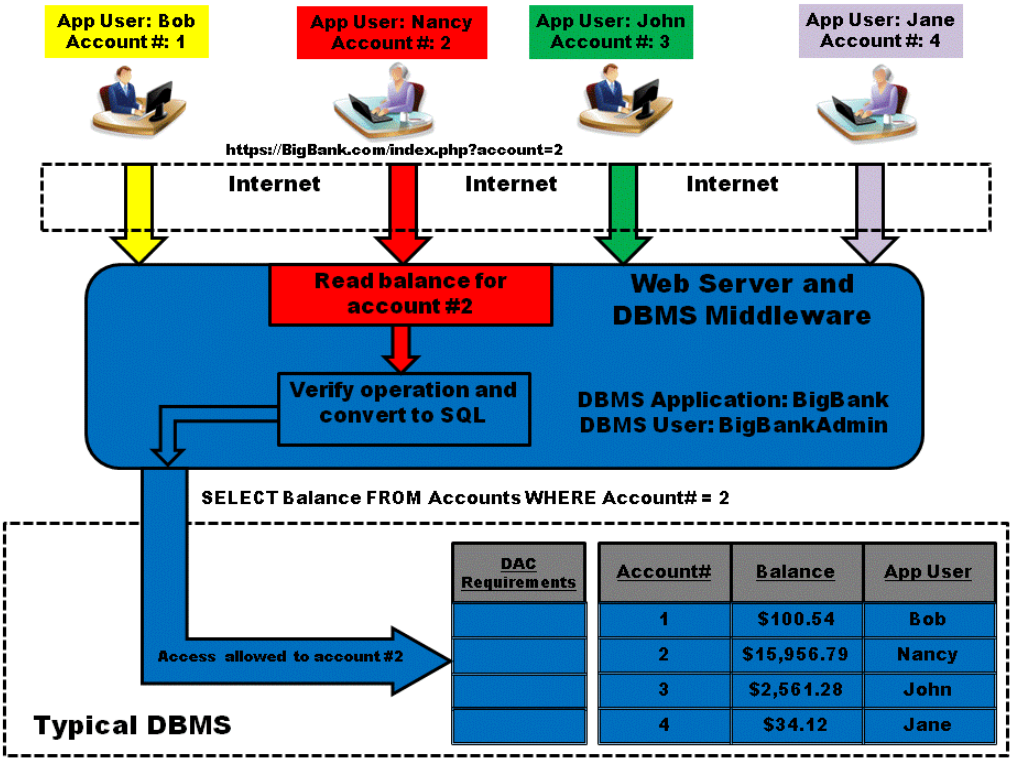

Figure 1: Typical RDBMS Middleware Architecture

The RDBMS application accepts operations from the web application, converts them into SQL, and submits them to the RDBMS server. The RDBMS application is typically an application custom made for the particular web application being developed. Typically, the RDBMS application is trusted to authenticate the application user, bind that authentication to the application user's session, and to ensure that no data is released or modified outside of that allowed for the authenticated application user. The RDBMS application connects to the RDBMS server as the RDBMS user and is trusted by the RDBMS server to perform any RDBMS operation that may be needed to satisfy any legitimate application user operation for any application user. Therefore, if the RDBMS application is compromised or bypassed, data from all application users is vulnerable.

The RDBMS server accepts operations from the RDBMS application in the form of SQL operations. The RDBMS server has no knowledge of application users. Therefore, a typical RDBMS server may not make any security decisions based upon the current application user.

SECURITY VULNERABILITIES

A typical web based solution that provides access to security critical, application user based information (e.g., online banking), often has security vulnerabilities in the RDBMS application. Fundamentally, this is because the RDBMS application has the following characteristics:

1. The RDBMS application is trusted to authenticate the application user (e.g., Internet banking user), typically using information stored in a database backend.

2. The RDBMS application is trusted to associate a currently authenticated application user (e.g., an authenticated Internet banking user) with the current application operation (e.g., read bank balance) in the RDBMS application. This is performed within a stateless HTTP Internet protocol. For example, if an operation arrives that asks to "read the bank balance", the application must construct SQL like:

```
SELECT Balance FROM Accounts WHERE Account#=2
```

The application must associate *Accoun#2* with the current operation and ensure that the user (e.g., *Nancy*) has been authenticated. This must occur in the context of a stateless HTTP protocol and concurrent RDBMS application operations from multiple users.

3. The RDBMS application must determine if the currently authenticated application user has the privilege to perform the current operation.

4. The RDBMS application is typically an ad-hoc solution specific to the web based solution that is needed. For instance, each bank may use their own, custom RDBMS and web applications. This results in more security related errors compared to a standardized, security targeted solution.

In short, the RDBMS application is vulnerable because it is trusted to perform all security checks related to the application user and the security critical application. Despite being the traditional, standardized point of security enforcement, the RDBMS server is not used for security enforcement because it has no concept of the application user and has no policy mechanism fined-grained enough to enforce access controls down to individual rows.

Example Use of Application Users

Next, an example is presented giving a typical, but simplified, use of the Application User mechanism. The example models a simple Internet banking application. The only functionality of the application is to allow a customer to read their current bank balance. The security objective is to utilize the Application User mechanism to permit access to a bank balance only when the corresponding, authenticated banking customer is requesting the balance through the middleware application. Figure 2 shows the architecture of the Internet banking solution.

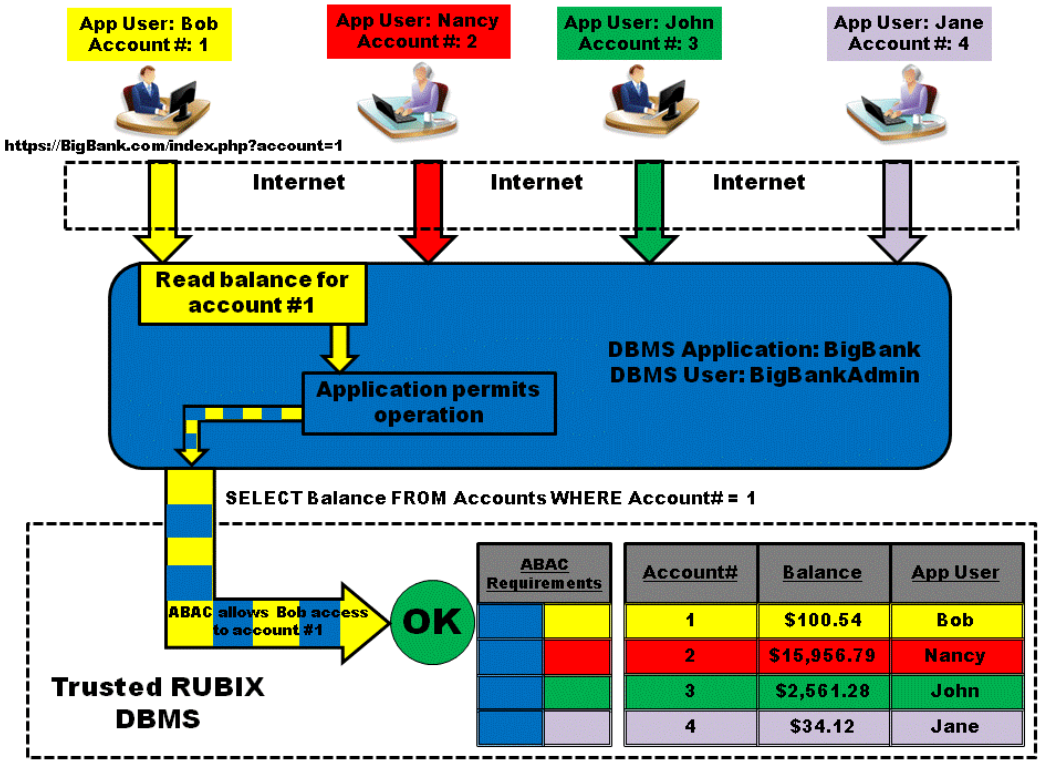

Figure 2: Internet Banking Solution using Trusted RUBIX Application Users

The top of the figure shows four Internet banking customers (i.e., Application Users) and their account numbers, *Bob* (#1, yellow), *Nancy* (#2, red), *John* (#3, green), and *Jane* (#4, purple). Each customer has a color which represents the Application User's ABAC permissions in the database. The Application Users connect to the banking application through the Internet.

The banking middleware application (e.g., Apache/PHP/ODBC application) has an Application name of *BigBank* in the Trusted RUBIX RDBMS. The Application Administrator is the RDBMS User *BigBankAdmin*. The blue color of the Application represents the ABAC permissions of the RDBMS User (i.e., the Application Administrator, *BigBankAdmin*) that executes the Application. This RDBMS User will connect to the Trusted RUBIX RDBMS and submit SQL operations on behalf of the Application Users.

Within the Trusted RUBIX database, shown at the bottom of the figure, is the *Accounts* table, with *Account#*, *Balance*, and *App User* columns. The *Accounts* table has four rows, one for each customer's bank account balance. Note that the color of each row corresponds to the color of the corresponding customer. Also note that the *App User* column is used to label each row with the creating Application User.

To the left of the *Accounts* table are the ABAC Requirements for each row. The colors shown represent the required permissions to access the row. Note that the color blue is required for each row, indicating that the authenticated RDBMS User *BigBankAdmin* must submit the SQL operations. Additionally, the permissions of the corresponding Application User is also required for each row. Thus, to access the first row, the permissions of the RDBMS User *BigBankAdmin* (blue) **and** the permissions of the Application User *Bob* (yellow) are required. For the second row, the permissions of the RDBMS User *BigBankAdmin* (blue) and the permissions of the Application User *Nancy* (red) are required, etc.

In the figure, Application User Bob has submitted a request to read the balance for account #1. Bob submits the request by submitting the following URL to the web server:

```
https://BigBank.com/index.php?acount=1
```

Note that the *account=1* component of the URL specifies which bank account balance is being requested. Assume that *Bob* has previously authenticated to the application and is set as the current Application User in the RDBMS. Following the flow of the operation down from the Application User *Bob*, the application accepts the operation and translates it into the corresponding SQL operation:

```
SELECT Balance FROM Accounts WHERE Account#=1
```

Because the Application User *Bob* and RDBMS User *BigBankAdmin* are authenticated and set in the database session, the SQL operation executes with the permissions of both users. This is reflected by the yellow and blue striped arrows representing the submitted SQL operation. As this matches the permissions required to SELECT the first row in the *Accounts* table, the SQL operation succeeds. The account balance of $100.54 will be returned to the Application User *Bob*.

Threats Mitigated by Application Users and ABAC

The Trusted RUBIX Application User mechanism in conjunction with the Attribute Based Access Control (ABAC) security enforcement mitigate several common security threats to data driven applications.

Threats fall into two broad categories: malicious user input to the RDBMS application and controlling the programming logic of the RDBMS application (i.e., application hijacking). In the first case, the hacker submits user input to the application that will cause it to execute SQL operations that violate the security requirements. In the second case, the hacker attempts to directly control the programming logic of the application to cause it to execute SQL operations that violate the security requirements. Note that in both cases, the vulnerability exists in typical RDBMSs because the RDBMS middleware application must execute with privileges sufficient to access **all** database objects necessary to satisfy **any** potential operation.

The Trusted RUBIX Application User mechanism removes these vulnerabilities by restricting the RDBMS middleware application to permissions sufficient to access **only** database objects that satisfy the **current** operation.

MALICIOUS USER INPUT

During a malicious user input attack the hacker submits input to the application that will cause it to execute SQL operations that violate the security requirements. Examples of this include SQL injection and URL manipulation.

In SQL injection attacks, the hacker attempts to exploit a vulnerability in the way the application constructs SQL command strings. As a simplified example, assume an application constructs the SQL command string by appending a given account number to the end of a WHERE clause, without checking that the input is in the proper form (e.g., a valid account number). Normal user input may request the balance of account '1' which would result in the proper SQL string:

```
SELECT Balance FROM Accounts WHERE Account#=1
```

This SQL operation would return the single, correct row. If, however, a malicious user requests the balance of account '1 OR 1=1' and the application does not properly sanitize the user input, the following SQL string may be constructed:

```
SELECT Balance FROM Accounts WHERE Account#=1 OR 1=1
```

This SQL operation would return all rows in the *Accounts* table. If the application displays this information to the user (i.e., the hacker) then a serious breach of security has occurred.

Trusted RUBIX would prevent this breach of security by limiting the permissions of the application to only allow access to the row for account #1. Thus, if the SQL command above was submitted, only the single, correct row would be returned.

In URL manipulation attacks, the hacker directly modifies the URL (e.g., in a web browser) in an attempt to control the SQL operations of the RDBMS application. As a simplified example, assume the application stores a user's account number as part of the visible URL during the session. So, if a user is operating on account #1, the following URL may be used:

```
https://BigBank.com/index.php?acount=1
```

The RDBMS application then takes the account variable value (1 in this case) and directly constructs the SQL command as:

```
SELECT Balance FROM Accounts WHERE Account#=1
```

Again, the SQL operation would return the single, correct row. A malicious user may attempt to exploit a vulnerability in the application by changing the URL to:

```
https://BigBank.com/index.php?acount=4
```

If the RDBMS application does not recognize that the request for the balance of account #4 violates the security objectives, then it may construct an SQL command as:

```
SELECT Balance FROM Accounts WHERE Account#=4
```

This SQL operation would return the balance for account #4 while the user's account is account #1. If the application displays this information to the user (i.e., the hacker) then a serious breach of security has occurred.

As before, Trusted RUBIX would prevent this breach of security by limiting the permissions of the application to only allow access to the row for account #1. Thus, if the SQL command above was submitted, only the single, correct row would be returned.

APPLICATION HIJACKING

During an application hijacking attack the hacker attempts to directly control the programming logic of an application to cause it to execute SQL operations that violate the security requirements. This may be done, for instance, by exploiting a buffer overflow vulnerability in the application program to cause it to execute arbitrary code. This arbitrary code could be constructed to execute any SQL operation the hacker desires. In this situation, the entire set of database objects the application has access to is accessible to the hacker, resulting in a serious security breach.

Trusted RUBIX dramatically reduces the vulnerability of such security breaches by limiting the permissions of the application to only allow access to database objects necessary to satisfy operations for the currently authenticated Application Users. If no Application Users are authenticated, then the hacker can access no database objects.

TYPICAL RDBMS VULNERABILITY EXAMPLE

This section describes a typical vulnerability found in RDBMS's that do not enforce a mandatory security policy on the users of the middleware application. Prior to reading this section, please become familiar with the sections titled "Example Use of Application Users" on page 7 and "Threats Mitigated by Application Users and ABAC" on page 8 of this document.

Figure 3 shows the architecture of a typical, vulnerable middleware application. It shows a simplified Internet banking application with four users. As in Figure 2, each user is color coded to represent the permissions it should have with respect to the application. The middleware application itself is colored blue to show the permissions of the RDBMS User executing the application.

Below the middleware application is the "typical" RDBMS. Note that all rows in the *Accounts* table are blue. This shows the permissions needed to access a row. In this architecture, unlike the Trusted RUBIX solution, the RDBMS User of the application may access all rows in the Accounts table, irrespective of which Application User is submitting the request. This presents a security vulnerability a hacker may exploit.

In this figure, a URL modification attack is shown. The Application User *Nancy* (account #2) has modified her URL to read the account balance for the account owned by *Bob* (account #1). This was accomplished by modifying Nancy's URL from

```
https://BigBank.com/index.php?acount=2
```

to

```
https://BigBank.com/index.php?acount=1
```

When this operation arrives at the middleware application, the program mistakenly does not verify the requested account number (#1) with the Application User requesting it (*Nancy*). It constructs the SQL command

```
SELECT Balance FROM Accounts WHERE Account#=1
```

and submits it to the RDBMS. Because the middleware application may access any row at any time, the SQL operation succeeds and the hacker receives the account balance of another customer (*Bob*).

With the Trusted RUBIX solution, the SQL operation would be denied because the Application User *Bob* was not authenticated and the Application User *Nancy* submitted an operation for account #1.

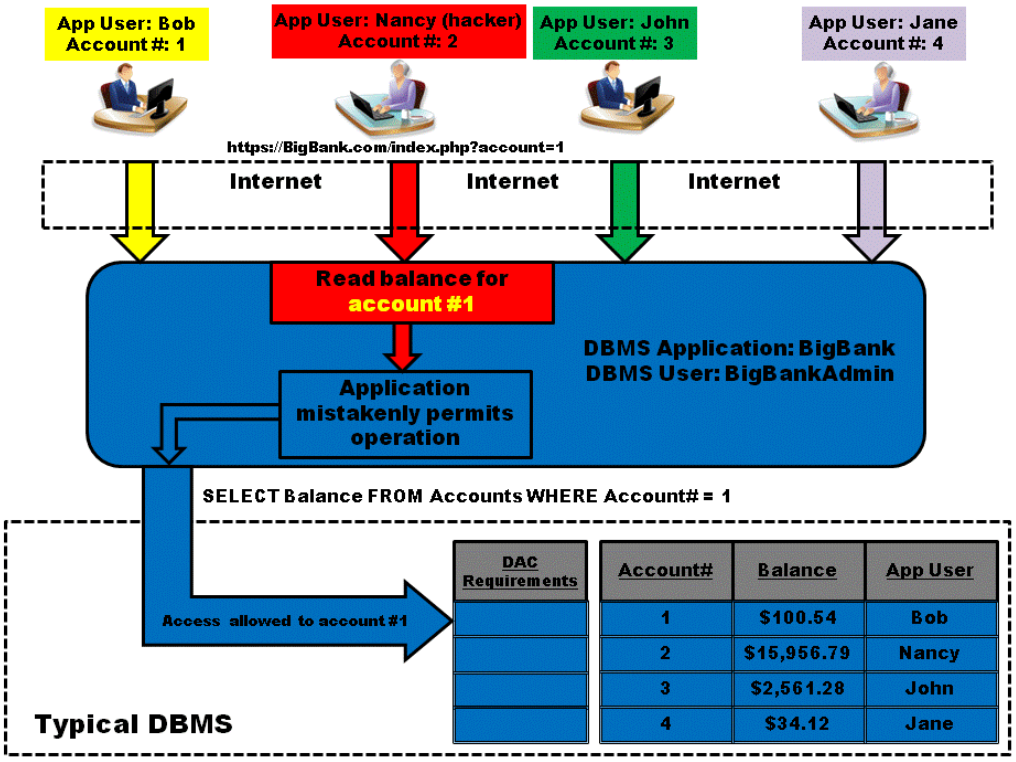

Figure 3: Typical RDBMS Vulnerability

TRUSTED RUBIX SOLUTION

This sections demonstrates how the Trusted RUBIX mandatory security policy on the users of the middleware application prevents a URL modification attack. Prior to reading this section, please become familiar with the sections titled "Example Use of Application Users" on page 7, "Threats Mitigated by Application Users and ABAC" on page 8, and "Typical RDBMS Vulnerability" on page 10 of this document.

Figure 4 shows the architecture of the Trusted RUBIX solution. It shows a simplified Internet banking application with four users. As in Figure 2, each user is color coded to represent the permissions it should have with respect to the application. The middleware application itself is colored blue to show the permissions of the RDBMS User (*BigBankAdmin*) executing the application.

Below the middleware application is the Trusted RUBIX RDBMS. Note that all rows in the *Accounts* table are color coded according to its corresponding Application User. The color shows the permissions needed to access each row. In this architecture the RDBMS User of the application may only access a row in the Accounts table if the corresponding Application User is authenticated.

In this figure, a URL modification attack is shown. The Application User *Nancy* (account #2) has modified her URL to read the account balance for the account owned by *Bob* (account #1). This was accomplished by modifying Nancy's URL from

https://BigBank.com/index.php?acount=2

to

https://BigBank.com/index.php?acount=1

When this operation arrives at the middleware application, the program mistakenly does not verify the requested account number (#1) with the Application User requesting it (*Nancy*). It constructs the SQL command

SELECT Balance FROM Accounts WHERE Account#=1

and submits it to the RDBMS. Because the ABAC security requirements to access the row for account #1 are for the RDBMS User *BigBankAdmin* (blue) **and** *Bob* (yellow) to be authenticated, the SQL operation fails and the hackers attempts are thwarted.

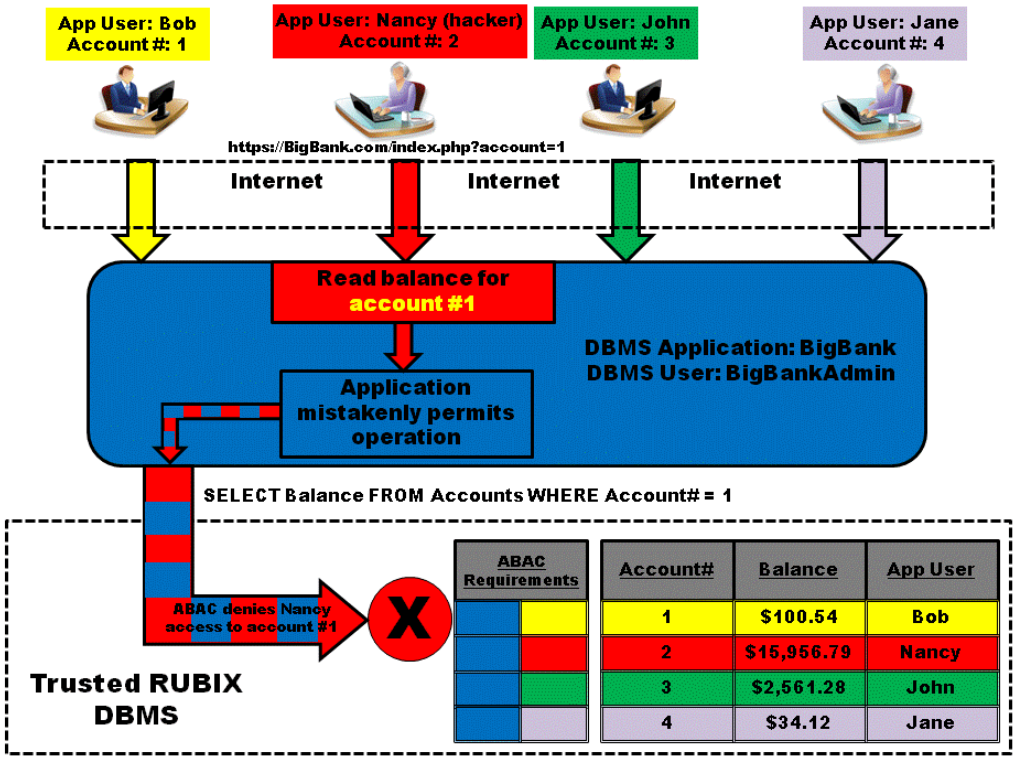

Figure 4: Trusted RUBIX Solution

Administrative Duties

The administrative duties for the application user mechanism are divided between the Security Administrator, the Database Administrator, and the Application Administrator. This was done to separate security relevant duties between roles, thus ensure all trusted administrative duties do not reside in one person.

Security Administrator Duties

The Security Administrator is responsible for those duties that directly impact system security.

Only the Security Administrator may:

- Create an Application Administrator (i.e., use the CREATE APPLICATION_ADMIN command).
- Drop an Application Administrator (i.e., use the DROP APPLICATION_ADMIN command).
- Drop an Application User that is not currently authenticated (i.e., use the DROP APPLICATION_USER command for a non-authenticated Application User). Note that an authenticated Application User may be dropped by the Application Administrator.
- Change the password for an Application User that is not currently authenticated (i.e., use the ALTER APPLICATION_USER SET PASSWORD command for a non-authenticated Application User). Note that the password for an authenticated Application User may be changed by the Application Administrator.

Database Administrator Duties

The Database Administrator is responsible for those duties that do not directly impact system security.

Only the Database Administrator may:

- Create an Application (i.e., use the CREATE APPLICATION command).
- Drop an Application (i.e., use the DROP APPLICATION command).
- Change the name of an Application (i.e., use the ALTER APPLICATION SET NAME command).
- Change the name of an Application User that is not currently authenticated (i.e., use the ALTER APPLICATION_USER SET NAME command for a non-authenticated Application User). Note that the name of an authenticated Application User may be changed by the Application Administrator.

Application Administrator Duties

The Application Administrator is responsible for normal duties that are required for the operation of the application. The Application Administrator is the RDBMS user that executes the named application.

Only the Application Administrator may:

- Set a named application as being current in the database session (i.e., use the ALTER SESSION SET APPLICATION command). Note that the Application Administrator must have been previously created by the Security Administrator.
- Authenticate an Application User to the RDBMS (i.e., use the AUTHENTICATE APPLICATION_USER command).
- Submit SQL operations that execute with the credentials of an authenticated Application User. Note that the SQL operation will actually execute with the credentials of both the Application User and the Application Administrator.
- Set an authenticated Application User as being current in the database session (i.e., use the ALTER SESSION SET APPLICATION_USER command). Note that the Application User must have previously authenticated to the RDBMS.
- Create an Application User (i.e., use the CREATE APPLICATION_USER command).
- Drop an Application User that is currently authenticated (i.e., use the DROP APPLICATION_USER command for an authenticated Application User).
- Change the password for an Application User that is currently authenticated (i.e., use the ALTER APPLICATION_USER SET PASSWORD command for an authenticated Application User).
- Change the name of an Application User that is currently authenticated (i.e., use the ALTER APPLICATION_USER SET NAME command for an authenticated Application User).

Related SQL Statements

The following SQL statements are used to manage the Trusted RUBIX Application User mechanism.

Application Statements

The following SQL statements are used to manage the Application.

CREATE APPLICATION

The CREATE APPLICATION command creates a named application in the RDBMS. RDBMS users may subsequently be designated Application Administrators for the Application using the CREATE APPLICATION_ADMIN command. These Application Administrators may then execute the RDBMS application program, connect to the database, and submit operations on behalf of Application Users. The CREATE APPLICATION command may only be used by those with the *rubix.app.create[.dbname]* authorization. By default, this authorization is given to the Database Administrator role.

As an example, to create an Application named “MyApp” issue the following SQL statement:

```
CREATE APPLICATION "MyApp"
```

DROP APPLICATION

The DROP APPLICATION command drops a named Application, all of its associated Application Administrators, and optionally all of the associated Application Users. The DROP APPLICATION command may only be used by those with the *rubix.app.drop[.dbname]* authorization. By default, this authorization is given to the Database Administrator role.

As an example, to drop an Application named “MyApp” and all of the associated Application Users issue the following SQL statement:

```
DROP APPLICATION "MyApp" CASCADE
```

ALTER APPLICATION

The ALTER APPLICATION command renames an existing Application. The ALTER APPLICATION command may only be used by those with the *rubix.app.alter[.dbname]* authorization. By default, this authorization is given to the Database Administrator role.

As an example, to rename the “MyApp” Application to “MyNewApp” issue the following SQL statement (note use of single and double quotes):

```
ALTER APPLICATION "MyApp" SET NAME = 'MyNewApp'
```

ALTER SESSION SET APPLICATION

The ALTER SESSION SET APPLICATION command sets a named Application as being current in the database session. The RDBMS user issuing this command must be an Application Administrator for the named Application. Once the current Application is set, the RDBMS application program may service operations

on behalf of Application Users that are associated with the Application. Additionally, the named Application becomes an attribute accessible by the ABAC mechanism.

As an example, to set the current Application in the session to “MyApp” issue the following SQL statement:

```
ALTER SESSION SET APPLICATION  = "MyApp"
```

Application Administrator Statements

The following SQL statements are used to manage Application Administrators.

CREATE APPLICATION_ADMIN

The CREATE APPLICATION_ADMIN command assigns a RDBMS user as an Application Administrator for a named Application. The RDBMS user may then execute the associated Application. The Application Administrator may set the Application as current in its database session, create Application Users that are associated with the Application, authenticate Application Users on behalf of the Application, and service operations on behalf of Application Users. The CREATE APPLICATION_ADMIN command may only be used by those with the *rubix.app.admin.creat[.dbname]* authorization. By default, this authorization is given to the Security Administrator role.

As an example, to designate the RDBMS user “MyAppAdmin” as an Application Administrator for the “MyApp” Application issue the following SQL statement:

```
CREATE APPLICATION_ADMIN APPLICATION  = "MyApp" USER = "MyAppAdmin"
```

DROP APPLICATION_ADMIN

The DROP APPLICATION_ADMIN commands removes the assignment of a RDBMS user as being an Application Administrator for a named application. The specified RDBMS user may no longer execute the associated Application. The DROP APPLICATION_ADMIN command may only be used by those with the *rubix.app.admin.drop[.dbname]* authorization. By default, this authorization is given to the Security Administrator role.

As an example, to remove the RDBMS user “MyAppAdmin” as an Application Administrator for the “MyApp” Application issue the following SQL statement:

```
DROP APPLICATION_ADMIN APPLICATION  = "MyApp" USER = "MyAppAdmin"
```

Application User Statements

The following SQL statements are used to manage Application Users.

CREATE APPLICATION_USER

The CREATE APPLICATION_USER command creates a named Application User for the Application that is currently set in the database session. The name for the new Application User must be unique within the Application. A numerical Application User ID is automatically generated which is unique amongst all

Application Users within the database. The authentication passphrase for the Application User is also given. The new Application User may subsequently authenticate and submit operations only to the current Application. Prior to using the CREATE APPLICATION_USER command, the Application in the current session must have been previously set by an Application Administrator using the ALTER SESSION SET APPLICATION command.

As an example, to create an Application User named "AppUser" with passphrase 'MyPassword" issue the following SQL statement (note that the current Application in the session determines which Application is associated with the Application User):

```
CREATE APPLICATION_USER "AppUser" WITH PASSWORD  'MyPassword'
```

DROP APPLICATION_USER

The DROP APPLICATION_USER command drops a named Application User for the Application that is currently set in the database session. Prior to using the DROP APPLICATION_USER command, the Application in the current session must have been previously set by an Application Administrator using the ALTER SESSION SET APPLICATION command. Additionally, the Application User being dropped must currently be authenticated or the RDBMS user must have the *rubix.app.user.drop[.dbname]*authorization. By default, the *rubix.app.user.drop[.dbname]*authorization is given to the Security Administrator.

As an example, to drop an Application User named "AppUser" issue the following SQL statement (note that the current Application in the session determines which Application is associated with the Application User):

```
DROP APPLICATION_USER "AppUser"
```

ALTER APPLICATION_USER

The ALTER APPLICATION_USER command changes the passphrase or the name for an existing, named Application User. Prior to using the ALTER APPLICATION_USER command, the Application in the current session must have been previously set by an Application Administrator using the ALTER SESSION SET APPLICATION command. Changing both the name and the passphrase is permitted if the Application User being changed is currently authenticated. Otherwise, those with the *rubix.app.user.alter.name[.dbname]* authorization may change the name and those with the *rubix.app.user.alter.password[.dbname]* authorization may change the passphrase. By default, the *rubix.app.user.alter.name[.dbname]* authorization is given to the Database Administrator role and the *rubix.app.user.alter.password[.dbname]* authorization is given to the Security Administrator role.

As an example, to change the name of an Application User named "AppUser" to "NewAppUser" issue the following SQL statement (note that the current Application in the session determines which Application is associated with the Application User; also note use of double and single quotes):

```
ALTER APPLICATION_USER "AppUser" SET NAME = 'NewAppUser'
```

AUTHENTICATE APPLICATION_USER

The AUTHENTICATE APPLICATION_USER command authenticates a named Application User using a given passphrase. Upon successful authentication the Application User's name and ID are set as current in database session and become attributes accessible by the ABAC mechanism. Additionally, the Application User is added to the pool of currently authenticated Application Users. Many authenticated

Application Users may reside in the pool but only a single authenticated Application User may be current at any given time. The Application User will remain authenticated until a failed authentication occurs or for the period of time defined by the Application's authentication timeout, as specified in the *rxconfig* configuration file. To end an Application User's authenticated session, the AUTHENTICATE APPLICATION_USER command may be issued with a null passphrase string. Prior to using the AUTHENTICATE APPLICATION_USER command, the Application in the current session must have been previously set by an Application Administrator using the ALTER SESSION SET APPLICATION command.

As an example, to authenticate an Application User named "AppUser" with passphrase "MyPassword" issue the following SQL statement (note that the current Application in the session determines which Application is associated with the Application User; also note use of double and single quotes):

```
AUTHENTICATE APPLICATION_USER = "AppUser" PASSWORD = 'MyPassword'
```

ALTER SESSION SET APPLICATION_USER

The ALTER SESSION SET APPLICATION_USER command sets the current session Application User name and ID in the current database session. Additionally, the Application User name and ID become attributes accessible by the ABAC mechanism If an Application User is set in the database session prior to the use of the ALTER SESSION SET APPLICATION_USER COMMAND, it will be removed. The Application User must currently reside in the authenticated Application User pool resulting from a previous use of the AUTHENTICATE APPLICATION_USER command. Prior to using the ALTER SESSION SET APPLICATION_USER command, the Application in the current session must have been previously set by an Application Administrator using the ALTER SESSION SET APPLICATION command.

As an example, to set the Application User named "AppUser" as current in the database session issue the following SQL statement (note that the current Application in the session determines which Application is associated with the Application User):

```
ALTER SESSION SET APPLICATION_USER = "AppUser"
```

SQL Statement Summary

Command	Use	Security Requirement
CREATE APPLICATION	Create an Application.	Database Administrator only.
DROP APPLICATION	Drop an Application, all associated Application Administrators, and (optionally) all associated Application Users.	Database Administrator only.
ALTER APPLICATION	Change the name of an Application.	Database Administrator only.
ALTER SESSION SET APPLICATION	Set an Application as current in the database session. Available as an attribute to the ABAC mechanism.	Application Administrator only.
CREATE APPLICATION_ADMIN	Create an Application Administrator that may execute and operate a named Application.	Security Administrator only.
DROP APPLICATION_ADMIN	Drop an Application Administrator.	Security Administrator only.

Command	Use	Security Requirement
CREATE APPLICATION_USER	Create an Application User.	Current Application must be set; Application Administrator only.
DROP APPLICATION_USER	Drop an Application User.	Current Application must be set; Application Administrator only; Application User must be authenticated or RDBMS user must be the Security Administrator.
ALTER APPLICATION_USER	Change the name or the passphrase of an Application User.	Current Application must be set; Application Administrator only; Application User must be authenticated or RDBMS user must be the Security Administrator (passphrase) or Database Administrator (name).
AUTHENTICATE APPLICATION_USER	Authenticate an Application User using a passphrase. Set as current in the database session. Available as an attribute to the ABAC mechanism.	Current Application must be set; Application Administrator only; Given passphrase must equal Application User's passphrase.
ALTER SESSION SET APPLICATION_USER	Set a previously authenticated Application as current in the database session. Available as an attribute to the ABAC mechanism.	Current Application must be set; Application Administrator only; Application User must be currently authenticated.

Retrieving Information Schema and Session Values

Information regarding Application Users, Applications, and Application Administrators may be obtained by quering the Trusted RUBIX Information Schema views. Information regarding the current Application User and Application as set in the database session may be obtained by selecting pseudo-literals values.

Information Schema Views

The Information Schema views may be queried to retrieve information about Applications, Application Administrators, and Application Users for a given database. For detailed information about the Information Schema and its associated tables and views please see the "Information Schema Guide."

As an example, to query all Applications in a database issue the following SQL statement:

```
SELECT * FROM system_catalog.info_schem.applications
```

THE APPLICATIONS VIEW

The *system_catalog.info_schem.applications* view contains information about all Applications created within the database. The columns of the *applications* view are:

app_name: The name of the Application. It has a character varying data type of length 128.

app_timeout: The timeout after which the authentication for Applications Users expire. It has an

integer data type.

THE APPLICATION_ADMINS VIEW

The *system_catalog.info_schem.application_admins* view contains information about all Applications Administrators created within the database. The columns of the *application_admins* view are:

app_name: The name of the application. It has a character varying data type of length 128.

app_admin: The name of the RDBMS user that has been designation an Application Administrator. It has a character varying data type of length 128.

THE APPLICATION_USERS VIEW

The *system_catalog.info_schem.application_users* view contains information about all Applications Users created within the database. The columns of the *application_users* view are:

app_name: The name of the application. It has a character varying data type of length 128.

app_user_name: The name of the Application User. It has a character varying data type of length 128.

app_user_id: The numerical ID of the Application User. It has a character varying data type of integer.

password: The SHA 256 hashed password and seed string for the Application User. It has a character varying data type of length 63.

Session Values

The current session values for the Application, Application User name, and Application User ID may be queried as pseudo-literals. For more information about using pseudo-literals please see the "Trusted RUBIX SQL Reference Guide."

As an example, to query the current Application User ID, issue the following SQL statement:

```
SELECT CURRENT_APPLICATION_USER_ID
```

THE CURRENT APPLICATION

The CURRENT_APPLICATION case-insensitive keyword may be used to query the current session Application (i.e., set by the ALTER SESSION SET APPLICATION command). If no Application is set in the current session then a NULL value is returned.

THE CURRENT APPLICATION USER NAME

The CURRENT_APPLICATION_USER case-insensitive keyword may be used to query the current session Application User name (i.e., set by the ALTER SESSION SET APPLICATION_USER or AUTHENTICATE APPLICATION_USER command). If no Application User is set in the current session then a NULL value is returned.

THE CURRENT APPLICATION USER ID

The CURRENT_APPLICATION_USER_ID case-insensitive keyword may be used to query the current session Application User ID (i.e., set by the ALTER SESSION SET APPLICATION_USER or AUTHENTICATE APPLICATION_USER command). If no Application User is set in the current session then a NULL value is returned.

Integration with RDBMS Application

The process of integrating the Trusted RUBIX Application User mechanism into an application requires only a small amount of special SQL programming. The following sections describe the steps needed to configure the Application User environment and to properly execute the Application.

RDBMS Configuration

To configure the application user tier the following steps are followed:

1. The Database Administrator creates a named Application (CREATE APPLICATION command). This Application corresponds to a RDBMS middleware application.

2. The Security Administrator creates one or more Application Administrators (CREATE APPLICATION_ADMIN command). An Application Administrator is a RDBMS user that is allowed to execute the middleware application and set that Application as current in the database session.

RDBMS Application Programming Logic

To service Application Users the following steps are followed by the Application:

1. The Application Administrator executes the middleware application, connects to the database, and sets the Application as being current (ALTER SESSION SET APPLICATION command).

2. The middleware application receives operations from application users. This includes creating new Application Users (CREATE APPLICATION_USER command).

3. The middleware application authenticates the Application User to the RDBMS (AUTHENTICATE APPLICATION_USER command). The RDBMS sets the Application User in the current database session.

4. The operations submitted by the Application User to the middleware application are mapped to SQL operations and submitted to the RDBMS.

5. The ABAC security policy of the SPM consults the current Application User and ensures only operations permitted for that Application User are allowed. It also ensures that rows created by the Application User are labeled as being created by the Application User.

The ABAC security policy of the SPM will ensure that the Application Administrator may not perform any operation restricted to an Application User unless that Application User is also currently authenticated. Thus, both the Application Administrator and Application User must be authenticated to the RDBMS for any such operation to succeed. Both identities are available as security attributes for ABAC security policies of the SPM.

Appendix A: Frequently Asked Questions (FAQ)

1) What about the needs of application domains, where there is a desire to control access to objects based upon the identity of users that are not recognized by the RDBMS and are only recognized by the application?

Typically such fine grained access control is left for the application code itself, rather than delegating this task to the RDBMS. However, this application aware approach has complicated enterprise security policies by exposing security policies to application programmers. The Trusted RUBIX Application User Concept was developed to address this problem.

2) In Trusted RUBIX terms, what is the relationship between a web user and an application user?

An application user is a "customer" of an RDBMS middleware application. An application user is recognized and typically authenticated only by the middleware application. An application user does not have an account on an RDBMS and its ID is typically meaningful only to the application middleware. For this reason, an RDBMS without the advanced security mechanism provided by Trusted RUBIX cannot enforce security on an application user. Thus, all security is typically enforced by the application.

A web user is an application user that resides on the web (Internet). An example would be an Internet banking customer.

The Trusted RUBIX solution only applies to users of Trusted RUBIX applications. Being on the Internet or not is immaterial. The TR solution is just as valuable for an architecture that is totally local (not on the web), but uses a Trusted RUBIX application.

3) What is the Application User/SPM mechanism and how does it work?

The Application User/SPM mechanism works as a pair to improve the SPM/ABAC based access control mechanism to control access to the row level based upon the ID of the application user, plus other attributes. Basically, it allows the authenticated ID of an application user to be used as an attribute in making ABAC policy decisions.

Trusted RUBIX provides the administration and authentication of the application user. Other than being an attribute in the ABAC mechanism, the ID of the application user has no meaning. It has no inherent privilege. Note the distinction between the application user and the Trusted RUBIX database user.

4) How does the Application User/SPM mechanism extend the MAC of the SPM to cover users of middleware applications?

It applies MAC even to the Trusted RUBIX application, the component usually relied upon to enforce "Internet user" security and the component frequently broken security wise. The Application User/SPM pair moves the task of limiting the operations of an application user under Trusted RUBIX MAC control, drastically reducing application vulnerabilities, such as application hijacking. A hacker who has gained complete control over an application can only get information from currently logged in users, but not from all users as with a typical RDBMS.

5) How does the Application User/SPM mechanism work to prevent SQL injection and URL tampering?

Most importantly, a proper configuration of the Application User/SPM prevents SQL injection

and URL tampering. It achieves this by removing the ability for the middleware application to perform any SQL operations that are not permitted by the current set of properly authenticated application users. This security enforcement is performed without using SQL operations and "where" clauses, as is typical when an application enforces security. Thus, any SQL operation submitted by the application, even one that was subjected to an SQL injection attack, will be constrained by the SPM's ABAC security policies.

6) What does integrating the Application User Concept with a cross domain environment provide?

Adding the application user concept to a cross domain environment moves the cross domain solution to one in which each separate domain can be separated per individual user (one policy with different users) to allow fine tuning the policy of each user. As opposed to making a decision solely upon the domain from which the user resides, the SPM in general can allow security decisions based upon a number of attributes, including but not limited to, the domain of the users. Trusted RUBIX does not need the Application User mechanism to perform the task, but the Application User mechanism adds one more attribute (application user) to the mix.

There are advantages of having the Application User concept integrated with MLS and/or TE. First, it allows the MLS policy to be enforced over users of a Trusted RUBIX application as opposed to MLS being enforced on an application as a whole. That is, the Trusted RUBIX RDBMS user running the application and the process label (or connection label) of the Trusted RUBIX application.

Thus, the following would be enforceable within the Trusted RUBIX server: A Trusted RUBIX application running at Top Secret, with an application user running at Unclassified - having multiple users at different labels running within the same Trusted RUBIX server.