Apex Security with Oracle Fine Grained Access Control

jim.schmidt@dbexperts.com

April 9, 2007

An Architecture for implementing Fine Grained Access Control within Web Applications and APEX

`version {$Revision: 1.1.2.9 $}`

Abstract

Database security involves restricting retrieval, update and deletion of records based on a security policy. A robust security policy provides a transparent security bastion at the database level that cannot be circumvented by developers intentionally or inadvertently.

Authentication is used to validate that a user is who he purports to be.

Authorization restricts the user to pre-defined functionality primarily as an enhancement to the Graphical User Interface by restricting displayed options to those sets of features for which the user has privileges.

Security employs Oracle Fine Grained Access to restrict data operations on the row or column level based on priviliges defined in the security schema.

Fine Grained Access control is the final authorization mechanism. It is enforced at the database level to ensure that no established policies are violated due to programming errors in the development of the Graphical User Interface. The fine grained access control that we will define will restrict all access to any table covered by the policy unless the operation is attempted by a database user connected as the owner of the schema. Consequently modifications will be necessary to the WebSite to use the procedures outlined below.

1 Overview

Terms

- *Application Server Context* An application server context, in this document is a set of services offered by an application server, a servlet container or APEX.

  A servlet container may have multiple contexts, each context consituting a web-site.

- *Database user.* A database user is the Oracle user that has established the connection to the database and will have an entry in dba_users.

- *Application user* An application user is defined by the application and Oracle has no intrinsic knowledge of this user.

- *Application Administrator* An Application Administrator is a special user with unlimited privileges within the application. This is an attribute of the user.

  Some tables may contain an ADMIN_READ_FLG column which indicates that records with this column with a value of 'Y' may only be read by an Administration User.

  Some tables may contain an ADMIN_UPDATE_FLG column which indicates that this record may not be changed or deleted by anyone who is not an Application Administrator.

  An application session is created by an application server, a servlet container or APEX, which we will generically refer to as an *application server* and is used within the

- *user session* A *user session* contains information an application session with user context information. When using database connection pooling such as when using APEX or a connection pool within your web sites it is important to be able to set the

2 Requirements

- Authentication A single authentication method will be used by all applications.
- Authorization All database access must be restricted to authenticated users unless the process is logged in as the schema owner.
- Authentication requires giving a username and password The password is hashed and compared to the hashed password for the user in ut_user
- The authenticated user can then set the session associated with the user but must do so using the same database connection that was used to do the authentication. This leaves us with the problem that we can't ensure that the user provides us with the session id at the same time. Any external application can provide a username, password and sessionid. Therefor we will allow APEX_PUBLIC_USER to make this in two calls.
- We need to be able to clean up the sessionids in the the authorization context
- we need make sure that nobody can iterate through the session id

3 Privileges

An *Application User* that is an Application Administrator can view or update any data in the database not otherwise restricted by Oracle Permissions.

4 Apex Security

Authentication is the process of determining that an *application user* has permissions to use the application based on a *user name* and *password*.

Authorization is the process by which APEX determines that the current *application user* is authorized to perform a given operation.

4.1 Authenticating a User

Create the item USER_NBR and ADMIN_USER_FLG on page 0.

Open the application in Application Builder and navigate to Shared Components - Security - Authorization Schemes - Application Express

4.2 Authorization Package

We assume a Oracle package with the following specification exists

```
create or replace package apex_authentication is

          function authenticate (
                    p_username in varchar2,
                    p_password in varchar2,
                    p_session_id in varchar2
          ) returns boolean;

          function get_user_nbr
               returns number;

          function get_admin_flg
               returns varchar2;

end apex_authentication;
```

The functions get_user_nbr and get_admin_flg perform destructive reads on package level variables. This ensures that a subsequent call to these methods don't reveal any data that was not associated with the APEX session that called the authenticate function. APEX shares database connections; as a result package level variables may be return to an APEX session that was not associated with the APEX session that set the package variables. The package

should also check the authorization time destroy the g_user_nbr and g_admin_flg if the subsequent call was not within 1 second of the authorization call.

In the Login Processing Box -¿ Authentication Function enter

```
return apex_authentication.authenticate;
```

In the Post-Authentication Process field enter.

The user_nbr is not used for authentication or authorization. It is used to mark records that are update or inserted

```
begin
        :user_nbr := apex_authentication.get_user_nbr;
        :admin_user_flg := apex_authentication.get_admin_user_flg;
end;
```

4.3 Authorization

Authorization is the process by which APEX determines that the current *application user* is authorized to perform a given operation.

4.3.1 Create an Authorization Package

```
create or replace package apex_authorization is
    function insert_allowed (
           p_session_id in varchar2,
           p_app_id in number,
           p_page_nbr in number
    ) return boolean;

    function update_allowed (
              p_session_id in varchar2,
              p_app_id in number,
              p_page_nbr in number
    ) return boolean;

    function delete_allowed(
            p_session_id in varchar2,
            p_app_id in number,
            p_page_nbr in number
    ) return boolean;

    function exec_proc_allowed (
           p_session_id in varchar2,
           p_app_id in number,
          p_page_nbr in number
    ) return boolean;

    function update_override_allowed(
             p_session_id in varchar2,
             p_app_id in number,
             p_page_nbr in number
    ) return boolean;
end apex_authorization;
/
```

The apex_authorization package checks with the security schema, please see8 to determine if the user associated with the given session has the requested privilige on the specified APEX page in the specified application.

4.3.2 Create an Authorization Scheme

Under Application - Shared Components - Security - Authorization Schemes

1. InsertAllowed

```
apex_authorization.insert_allowed(p_session_id => :session_id,
                                                    p_app_id => :app_id,
                                                    p_page_id => : app_page_id);
```

2. UpdateAllowed

```
apex_authorization.update_allowed(p_session_id => :session_id,
                                                    p_app_id => :app_id,
                                                    p_page_id => : app_page_id);
```

3. DeleteAllowed

```
apex_authorization.delete_allowed(p_session_id => :session_id,
                                                    p_app_id => :app_id,
                                                    p_page_id => : app_page_id);
```

4. ExecProcAllowed

```
apex_authorization.exec_proc_allowed(p_session_id => :session_id,
                                                    p_app_id => :app_id,
                                                    p_page_id => : app_page_id);
```

5. UpdateOverrideAllowed

```
apex_authorization.update_override_allowed(p_session_id => :session_id,
                                                    p_app_id => :app_id,
                                                    p_page_id => : app_page_id);
```

Choosing this Authorization Scheme results in the Apply Changes button showing up only when the function associated with it returns TRUE.

These functions return TRUE or FALSE depending on the whether the Actions are allowed for the User Logged in.

Hiding or showing the Buttons on an APEX page is not fool proof. If the developer fails to assign an Authorization Scheme to a Button on a page, users will be able to execute the actions associated with it even if they are not Authorized to do so. To prevent this, we have added a second layer of security that is enforced at the database level. This is used by assigning Insert, Update and Delete policies to every table in the schema.

4.3.3 Conditional Control Display

The user's browser window should not display controls that the user is not authorized to use.

In order to prevent a button from displaying if the user is not authorized to use the button

1. select the item from the page edit view in APEX.
2. click on the Authorization link
3. choose the appropriate authorization scheme from the list defined above for the operation in question. For example if the button deletes the selected record, choose the *DeleteAllowed* authorization scheme.

The apex_authorization package implementation, *package body* will return false if the user does not have the authorization to perform the operation based on the rules store in the database and represented in the Entity Relationship diagram at BROKENLINK.

For example if a user may not update a record, the update button should not be displayed.

4.3.4 Implementing the Authorization Scheme

Once created, an Authorization Scheme can be assigned to Items/Buttons/Pages within the Application to make them appear if the user is authorized by the authorization scheme. Application User

5 Virtual Private Databases

http://www.databasejournal.com/features/oracle/article.php/3644956

The introduction of the DBMS_RLS package in Oracle 9i offered an excellent alternative to the custom-written view implementation of security. As its name implies, DBMS_RLS allows a DBA to enforce row level security against specific tables in the database. Whenever a row is read, added, modified or deleted, Oracle applies fine grained access control (FGAC) rules that insure the rows values met the strictures of that predefined security policy.

The security policy enforces these restrictions by adding a hidden predicate to each query or DML statement that attempts to access the data. For example, if a query attempts to access a row, and the security policy determined that the user had insufficient permission to access it, then Oracle filtered the row from the querys result set. On the other hand, if a DML operation attempted to process the row, and the security policy showed that the user was limited from accessing the row, Oracle blocked the operation against the row.

5.1 Using the DBMS_SESSION Interface to Manage Application Context in Client Sessions

The DBMS_SESSION interface for managing application context has a client identifier for each application context. In this way, application context can be managed globally, yet each client sees only his or her assigned application context. The following interfaces in DBMS_SESSION enable the administrator to manage application context in client sessions:

SET_CONTEXT
CLEAR_CONTEXT
CLEAR_ALL_CONTEXT (can also be used with session-based application context)
SET_IDENTIFIER
CLEAR_IDENTIFIER

The middle-tier application server can use SET_CONTEXT to set application context for a specific client ID. Then, when assigning a database connection to process the client request, the application server needs to issue a SET_IDENTIFIER to denote the ID of the application session. From then on, every time the client invokes SYS_CONTEXT, only the context that was associated with the set identifier is returned.

http://download-west.oracle.com/docs/cd/B19306_01/network.102/b14266/apdvpoli.htm#i1009723

5.2 Column-Level VPD with Column-masking Behavior

If a query references a sensitive column, then the default behavior of column-level VPD restricts the number of rows returned. With column-masking behavior, which can be enabled by using the sec_relevant_cols_opt parameter of the DBMS_RLS.ADD_POLICY procedure, all rows display, even those that reference sensitive columns. However, the sensitive columns display as NULL values.

To illustrate this, consider the results of the sales clerk query, described in the previous example. If column-masking behavior is used, then instead of seeing only the row containing the details and Social Security number of the sales clerk, the clerk would see all rows from emp, but the ssn column values would be returned as NULL. Note that this behavior is fundamentally different from all other types of VPD policies, which return only a subset of rows. What about context_sensitive policy_type

http://download-west.oracle.com/docs/cd/B19306_01/network.102/b14266/apdvpoli.htm#i1009723

5.3 Using the CLIENT_IDENTIFIER Attribute to Preserve User Identity

The CLIENT_IDENTIFIER, a predefined attribute of the built-in application context namespace, USERENV, can be used to capture the application user name for use with global application context or it can be used independently. When used independent of global application context, CLIENT_IDENTIFIER can be set with the DBMS_SESSION interface. The ability to pass a CLIENT_IDENTIFIER to the database is supported in OCI and thick JDBC.

When CLIENT_IDENTIFIER is used with global application context, it provides flexibility and high performance for building applications. For example, suppose a Web-based application that provides information to business partners has three types of users: gold partner, silver partner, and bronze partner, representing different levels of information available. Instead of each user having his own session set up with individual application contexts, the application could set up global applications contexts for gold partners, silver partners, and bronze partners. Then, use the CLIENT_IDENTIFIER to point the session at the correct context in order to retrieve the appropriate type of data. The application need only initialize the three global contexts once and use the CLIENT_IDENTIFIER to access the correct application contex

http://download-west.oracle.com/docs/cd/B19306_01/network.102/b14266/apdvcntx.htm#i1009024

6 Oracle Fine Grained Access

The mechanism in the preceeding section conditionally allows users priviliges by selectively displaying controls that allow certain operations. However, if the developer fails to apply the Authorization Scheme to an Apex component there is a potential for security to be compromised. Consequently we employ Oracle Fine Grained Access control to provide a transparent security bastion.

6.1 Requirements

We stipulate the following requirements:

- No restrictions should be placed on a user connected to the database as the GUS user.
- No user other than GUS should be able to view or modify any of the contents of any of the access controlled tables unless he has been authenticated using the authentication package.
- Ability to view and modify information based on user/manufacturer privilege association

```
PROCEDURE set_mfr_context (
 p_session_id             IN  VARCHAR2,
 parm_org_nbr_mfr    IN  PLS_INTEGER ) ;
```

- Ability to view and modify information based on user/distributor privilege association

```
PROCEDURE set_dst_context (
 p_session_id             IN VARCHAR2,
 parm_org_nbr_dst    IN  PLS_INTEGER ) ;
```

6.2 Define the Security Policy Functions and Procedures Specifications

```
create or replace
PACKAGE security_context IS
--
--   The following functions set  the associated security context
--
--
PROCEDURE set_session_context (
 p_session_id             IN  VARCHAR2 ) ;

PROCEDURE set_mfr_context (
 p_sesion_id              IN  PLS_INTEGER,
 parm_org_nbr_mfr    IN  PLS_INTEGER ) ;

PROCEDURE set_dst_context (
 p_session_id            IN  PLS_INTEGER,
 parm_org_nbr_dst    IN  PLS_INTEGER ) ;

--
--   The following functions read from the associated security context and apply constraints on
--    the database object access by appending to the predicate on the associated SQL statement
--
FUNCTION get_insert_context (
 parm_schema_nm     IN  VARCHAR2,
 parm_object_nm     IN  VARCHAR2 )
RETURN VARCHAR2 ;

FUNCTION get_update_context (
 parm_schema_nm     IN  VARCHAR2,
 parm_object_nm     IN  VARCHAR2 )
RETURN VARCHAR2 ;
```

```
FUNCTION get_org_nbr_mfr_context (
 parm_schema_nm     IN  VARCHAR2,
 parm_object_nm     IN  VARCHAR2 )
RETURN VARCHAR2 ;

----------------------
--
--  Tables with an ADMIN_READ_FLG or ADMIN_UPDATE_FLG column will need an additional access control to ens
--   records are not viewed or changed except by an administrator
--
----------------------

FUNCTION get_select_context_admin_opt (
 parm_schema_nm     IN  VARCHAR2,
 parm_object_nm     IN  VARCHAR2 )
RETURN VARCHAR2 ;

FUNCTION get_insert_context (
 parm_schema_nm     IN  VARCHAR2,
 parm_object_nm     IN  VARCHAR2 )
RETURN VARCHAR2 ;

FUNCTION get_update_context_admin_opt (
 parm_schema_nm     IN  VARCHAR2,
 parm_object_nm     IN  VARCHAR2 )
RETURN VARCHAR2 ;

FUNCTION get_delete_context (
 parm_schema_nm     IN  VARCHAR2,
 parm_object_nm     IN  VARCHAR2 )
RETURN VARCHAR2 ;

END security_context ;
```

6.3 Creating policies

To create security policies for a Table, grant the following privileges to the user logged in as SYS.

```
GRANT EXECUTE ON dbms_rls TO gus ;
GRANT CREATE ANY CONTEXT TO gus ;
GRANT EXECUTE ON dbms_session TO gus ;
GRANT ALTER SESSION TO gus ;
```

6.4 Create a context

Once the required grants are available, create a context as follows.

```
CREATE OR REPLACE CONTEXT gus USING security_context ;
```

This statement creates a context named gus and ensures that access to the context may only be made by the security_context package defined above.

This package need not exist when creating the context.

A context is essentially a named set of name value pairs.

The Next step is to create a policy on a table. This is done as follows.

```
BEGIN
        dbms_rls.add_policy (
                object_schema        =>USER,
                object_name            =>'ITEM',
                policy_name            =>'INSERT_POLICY',
```

```
                function_schema =>USER,
                policy_function          =>'security_context.get_insert_context',
                statement_types =>'insert' ) ;
END;
/
```

This statement creates a policy named INSERT_POLICY on the table ITEM for the INSERT operation and enforces it using the Function

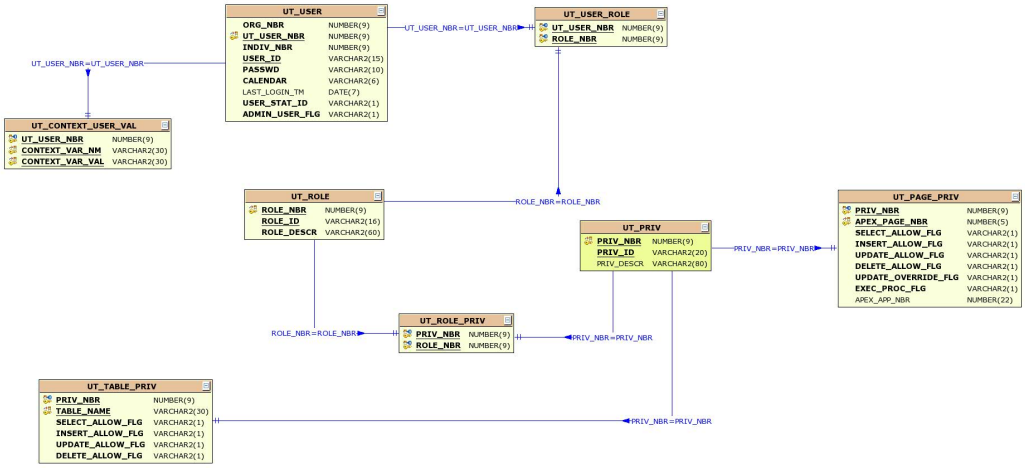
`security_context.get_insert_context.`

6.5 Implement Security Policy

7 Related Commands

grant exempt access policy to god;

8 Defining Roles



9 More Infomation

http://www.proligence.com/nyoug_fgac.pdf
http://rjh.keybit.net/oracle/Chapter%2021.htm
http://hosteddocs.ittoolbox.com/LC100705.pdf
http://orafaq.com/node/58
http://www.oracle.com/technology/oramag/oracle/05-jan/o15security.html