ArcIMS and Oracle row level security

Introduction

The project we are going to talk about was initiated by the Metropolitan Transportation Commission (MTC), the San Francisco Bay Area's metropolitan planning organization. The purpose of the project was to build a web-based system that provides access to traffic signal data to the general public and data maintenance tools to the agencies that own and operate the traffic signals.

Both public users and the agencies are using the same web based interface to query data. Results of the queries are presented in tabular form or as interactive maps. The agencies also have access to the content management system (CMS), which provides tools for data maintenance and user access control.

The following are some of the requirements identified during the analysis process, which shaped our solution:

- User authentication/authorization has to be handled by RDBMS to avoid unauthorized data access
- Users have to have an ability to grant/revoke permissions at record (feature) group level
- The system has to ensure limited access to the sensitive data
- The system has to provide personalized map content depending on the requestor's data access permissions
- Data owners and maintainers have to be able to grant/revoke access permissions to their data by themselves

We encountered certain limitations of the ArcIMS user authorization methods while working on the project; therefore we would like to share our experiences with overcoming these limitations, especially using user authentication/authorization data (including non-encrypted passwords) in XML files or database tables.

Proposed solution

Each agency in the system owns and maintains a group of traffic signals records. The agency, according to the requirements, should be able to control (grant and revoke permissions) their data by themselves.

At the first sight, one of the easiest solutions would be to create multiple spatial views and ArcIMS mapping services for every agency in the geodatabase and to control access to the services using ACLs. In this case, each user could be granted permissions on the part of the mapping services. At the same time such solution would be difficult to maintain due to the large numbers of users and mapping services in the system.

After the research, we decided to use Oracle row level security solution (known as Virtual Private Database), which comes as part of Oracle Enterprise Edition. Using this solution, the database can be set up the way that users must be authorized to access data not only at a table but at the each record level as well; the same setup would work for both attribute and spatial data.

ArcIMS user authorization

ArcIMS provides few methods to control user access to the map services. It can be implemented using Access Control Lists (ACL) as a text file (XML) or database based solution. As an alternative, web server security can be used to restrict users from accessing certain web applications. Such access control works fine until one needs something more granular than access control at the application or mapping service level. It doesn't work good enough when there is a need for row-level access control with multiple users accessing the same data or in the cases when authentication/authorization is managed by RDBMS and there are no passwords stored in a user's profile for security reasons.

Oracle row level security (Virtual Private Database)

Oracle row level security (Private Virtual Database) comes as a part of Oracle Enterprise Edition. It is based on custom triggers and functions, which set session context and define user access to the data on a row level. First, login trigger sets application context. Policy function is being called every time when table or view is queried or altered. Which policy function will be used depends on the context. The predicate returned by the function is added to the where clause of the original SQL statement. This way, the policy function defines which rows are returned to the user as a result of the query. This mechanism works transparently to the user (or application) and doesn't require any coding on the client side – everything is handled by the RDBMS itself. Such approach simplifies maintenance, development, and ensures that no ad-hoc queries or tools/applications would be able to get unauthorized access by bypassing application side authorization logic.

Implementation

Two main modules were developed – RDBMS authorization module and application server side module.
We had to make some changes in the initial database design to accommodate authorization information. These changes were minor and affected the agency (data owner) profile table only. An additional table was created to store granting data. The agency profile was updated to store information about data that were made accessible for the public in the form of several flags. The grants table was created as many-to-many relationship resolution table between agencies' users and the agencies that granted them permission to access their data.
The second step was to develop policy functions. These functions are checking users permissions and returning data:

- owned by the agency the user belongs to
- owned by agencies which granted permission to access their data to the user
- owned by agencies which granted access to their data (or part of data which is being queried) to the public.

This way, after setting up the database, we developed system, which allows users to login to the database, to query it using any tool or application, and to see only the data they are allowed to. To illustrate how it works we will use following scenario:

- Agency "Alameda" has 58 signal records in the database, "Alameda County" – 79, "Oakland" – 609.

- Row level access is enforced on the signal table.
- Alameda has user called “Alameda1”, Alameda County – “AlamedaCnty1”, Oakland – “Oakland1”.
- Agency “Oakland” granted access to its data to “Alameda County” user “AlamedaCnty1”.
- There are no agencies in the system that granted access to their data to the public.

When user Alameda1 tries to get number of the signal records from the database using SQL Plus (“SELECT COUNT(*) FROM LOCATION”) – result will be 75. If user Oakland1 does the same – result will be 609. In the case of AlamedaCnty1 user – result will be 746.

We created ArcSDE (spatial) view to provide access to the spatial data. The view is based on the location table (along with separate location point feature class in the geodatabase). This way, the content of the view always depends on who is asking for the data (which user account was used to login).

Now that we have the database set up and ready, let’s focus on the presentation side. We developed the web application using ArcIMS Java connector, JSP, Java beans and Java servlets. One can look for samples of such applications on the Internet, and there is also some information (regarding ArcIMS Java connector) provided as a part of the ArcIMS documentation. We will focus on a few issues related to the subject of our presentation.

Usually, ArcIMS Java connector based applications (providing simple map viewer functionality) employ the ArcIMS mapping service, configured using an axl file. User names and passwords used to access geodatabases are stored in the file itself (password can be stored encrypted). As a result, we have an “application gateway” which performs user authorization based on ACL and uses a single user account to access data. Such an approach, with “hardcoded” usernames, didn’t work for us because all users were supposed to access spatial data using their own accounts.

The solution we found is based on the ArcIMS Java (and ActiveX as well) connector’s ability to dynamically add/remove workspaces and layers to/from the existing mapping service. We use general mapping service with “hardcoded” username/password for reference map. A layer with user specific data (which is pointing to aforementioned spatial layer) is being added/removed dynamically for each user. At the beginning of the request, right after the connection to ArcIMS is established and map object is initialized, a new ArcSDE workspace is created using the current user username/password. After that, a new feature layer is created and added to the map object layers collection. The renderer of this layer is also set up dynamically and could depend on user role/privileges/etc. if needed (it is not implemented in current version of the application).

As any other solution, this one has its own advantages and disadvantages. It is important to understand them before making the decision because the approach we had taken may not work for you.

Advantages:

- *More secure user authentication and authorization*. This approach provides a flexible authorization mechanism and at the same time moves all authentication handling to Oracle RDBMS. There are no more XML files or database tables with usernames and unencrypted passwords.
- *Simplified user management.* All user authentication/authorization information is stored in Oracle RDBMS. Most of user management tasks can be done using standard Oracle or third party tools or web applications.
- *Flexible data access rules and simpler geodatabase design.* Ability to define access rules at the record level provides more flexibility and easier geodatabase design when there is a need for fine grained data access control. Table (feature class) level control (using grants) or column level control (using views) may not be the best solution in such cases.

Disadvantages:

- *Overhead of reconnection/dynamic layer.* In our case, processing of each request involves reconnecting to the geodatabase and dynamically adding a map layer. This overhead can be eliminated in cases when session state between requests is maintained as session variable and stored on the application server side.
- *Overhead created by policy functions.* Additional functionality never comes for free. Still, in our case the geodatabase isn't so big and we do not have very high loads to notice this overhead.
- *Custom solution needed.* There is no easy way to use row level security with standard out of the box solutions or solvers provided as the part of ArcIMS installation. Oracle RDBMS side coding is also required.
- *Management of large numbers of users.* There can be lots of registered users in publicly accessed websites. All of these registered users would have their own accounts in Oracle RDBMS and it may become really hard to manage all these accounts.

Conclusions

ArcIMS provides out of the box solutions based on ACLs for user authorization at a map service level. In many applications it is enough. However, if you need more a flexible mechanism for authorization at the more detailed level – you should think of a custom solution. One of such solution may be based on establishing Oracle row level security (Virtual Private Database). In this case you will have both user authentication and authorization handled by Oracle RDBMS and flexible data access rules enforced by the RDBMS.