Enterprise Role-based Security De-Mystified

Bill Hulsizer – April 2005

Role-based security is simple in theory. Users are defined and assigned a Business Role, which corresponds to job functions they perform. Business Roles are assigned the Application Roles they need. Application Roles are usually access to specific menu options within an application, but can also be access to do data mining in an enterprise data warehouse or access to any other application. Application Roles are assigned the Data Roles necessary to access the data they provide to users. Data roles can be select, update, insert, delete or administrative roles for individual data stores and up to enterprise-wide data warehouses. Figure 1 below depicts this architecture.

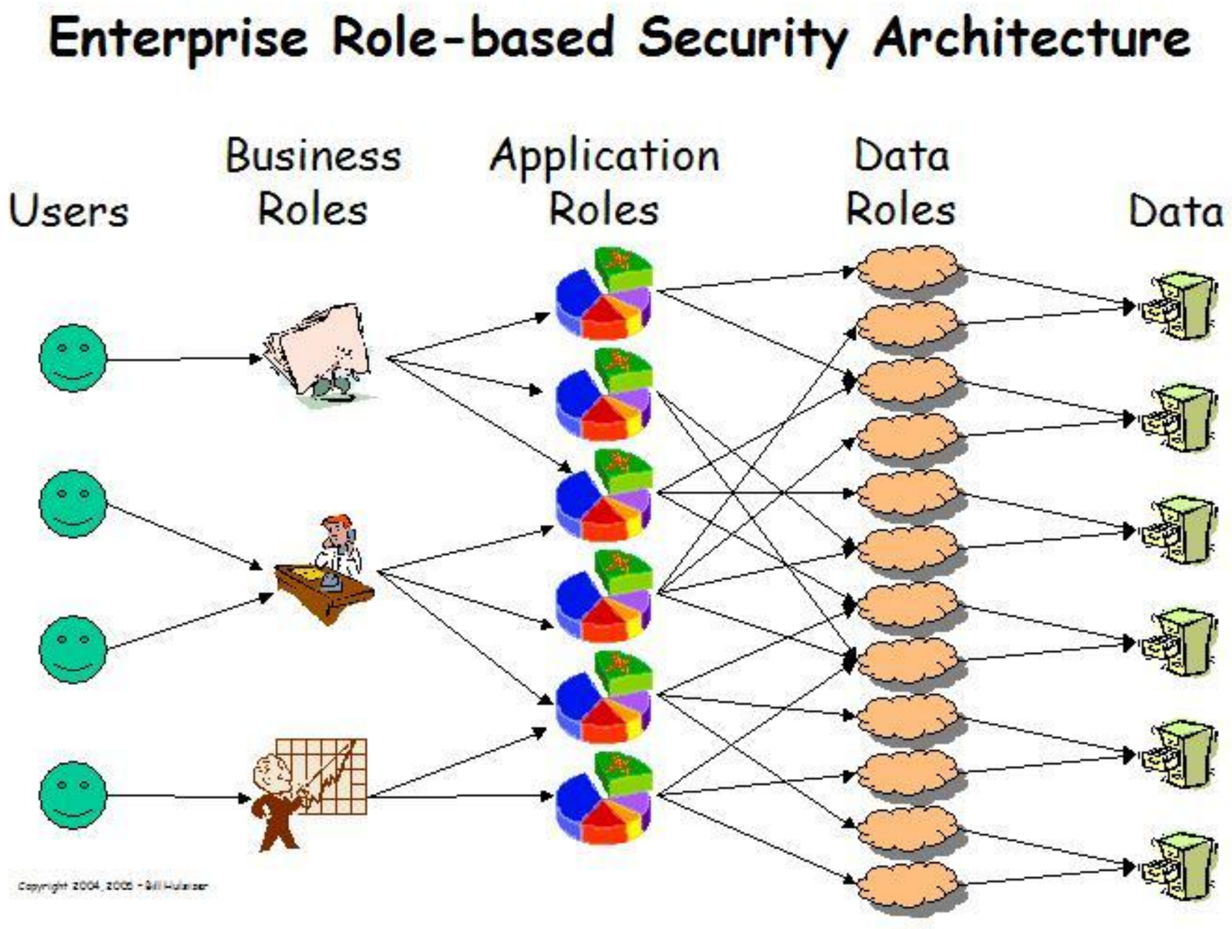
Enterprise Role-based Security Architecture
Users
Business Roles
Application Roles
Data Roles
Data
Copyright 2004, 2005 - Bill Hulsizer

Figure 1

The Business, Application and Data Roles in Figure 1 are all operating system security groups. On Windows, they are Windows security groups, on IBM mainframes they are RACF security groups, on AS400s they are OS400 security groups, etc. Creating users, assigning users Business Roles, assigning Business Roles their Application Roles and assigning Application Roles their Data Roles are all done using standard operating system user management tools. Applications and databases verify role membership using operating system interfaces (OSIs), which happen to function identically across all major commercial operating systems.

Operating System Interfaces

OSIs exist for every major commercial operating system to provide two basic role membership functions. The first OSI provides a recursive (nested) list of all operating system security groups for a given user ID. This contains the user's one business role and all related application and data roles. An application can retrieve this list once or periodically and search it to determine role membership as needed. The second OSI provides a yes/no answer when passed a security group and user ID. If the user ID is a member of the security group, then the answer is yes, otherwise the answer is no. This is how DB2, Oracle and SQL Server relational databases verify security for accessing data within their databases. Internally within the databases, GRANT and REVOKE commands are used to grant specific authorities to roles. These roles are really operating system security groups. When a person executes an SQL command to retrieve or change data, the database manager gets a list of roles which have been granted the internal security to do this. It then uses the second OSI above to determine if the user is a member of any of these roles. For example, to insert a new row into a table, the user would have to be a member of a group that has been granted insert security for that table. The RDBMS executes the OSI once per role that has been granted the security until it finds one the user is a member of. (DBA note: This is why reducing the number of roles granted internal database authority is important.) Applications can be written to use these OSIs similar to how databases currently use them.

Role-based Application Security

Applications are designed with control points where security is checked to verify the user is allowed to proceed. Each control point should be assigned an application role, which is the name of an operating system security group. Members of this operating system security group are allowed to pass that control point. Applications use the OSIs discussed above to verify membership. Some benefits of this architecture include:

- Application and data security are externalized to the operating system, the most secure environment possible.
- Application and data security can be centrally administered using operating system security techniques, reducing training.
- Using enterprise-wide operating system security management tools, operating system, application and data security can be centralized using one administrative tool, providing further cost savings and tighter security.

Role-based Data Security

There are many sources of data on computers today. Relational databases like DB2, Oracle and SQL Server, as well as file-based technologies like B-Trieve and C-ISAM. Relational databases have their own internal security that is assigned using the GRANT and REVOKE commands, as discussed above. Non-relational data sources are secured directly by the operating system using, for example, file and folder permissions for Windows or RACF dataset permissions for IBM mainframes. In Oracle and SQL Server, users and roles must be defined within the database, but they should always be defined as "external". This means they are really operating system users and security groups. DB2 does not have internal users or roles, because DB2 users and roles are always externalized to the operating system.

Data Roles

Some relational databases limit Data Role names to only eight characters in length. This is a severe limitation, so Data Role names need to be chosen very carefully. Separate Data Roles need to be created for the environment (test vs. production) and the type of access needed (select, insert, update, delete, admin). The name must also signify that the role is a database role and not some other kind of role. The following naming convention works very well and will be used for the purpose of this discussion:

Node	Description
1	***DRole*** - All Data Role names start with *DRole* to distinguish them from other operating system security groups. No other type of security group name should start with *DRole*.

2 **Environment** - *Devel* for development test, *Syst* for system test, *User* for user test, *Model* for model, *Prod* for production. Other names can be used for other environments. *Test* should never be used to avoid confusion.

3 **Security Level** – *System* for system-level, *Dbase* for database-level, *Schema* for schema-level and *Table* for table-level security. Each of these is discussed in more detail below.

4 **Access Type** – Table access types are *Select, Update, Insert, Delete* or *All* (for all four). Normally only *Select* and *All* are needed, but *Update*, *Insert* and *Delete* should be available for use as needed. Administration access types can be any internal types needed, like *SysAdm, DbAdm, SysOpr, etc.*

5 **Security Object** - The system, database, schema or table name.

Using this naming convention, an example of a Data Role name for the production environment at database-level with all authority in the accounts receivable database AcctRecv is DRole_Prod_Dbase_All_AcctRecv. Efficiency is important for data security. This means users should be in as few Data Roles as practical. And authority for each table should be granted to as few Data Roles as practical. Accomplishing this can mean using a combination of database-, schema- and table-level security.

System-level security

System-level security is for administrative roles like system administrator or system operator. A system would be an instance for open systems databases or a subsystem for DB2 using the z/OS operating system. An example of a Data Role for the production environment at system-level for granting system administrator privileges for the DB2 subsystem DB2P is DRole_Prod_System_SysAdm_DB2P. This role would have SYSADM privileges granted to it and anyone requiring that security would be a member of that role.

Database-level security

Database-level security assumes that everyone who needs access needs the same security for all tables in the database. This is particularly true for secure user IDs used by secure applications when they connect to the database. These IDs usually need select, insert, update and delete authority for all tables the application accesses. It is

OK to allow this because these applications are audited and have internal security to prevent unauthorized actions by users. If an entire database is being used by a secure application using a secure user ID, then all tables in the database need to have select, insert, update and delete authority granted to one database-level Data Role. Using a prior example, all tables in the accounts receivable database AcctRecv need to have select, insert, update and delete authority granted to Data Role DRole_Prod_Dbase_All_AcctRecv. Since this Data Role provides a high level of authority, its members should be highly restricted and user passwords highly controlled. Database-level security also works well for data warehousing applications. Typically all users need inquiry authority to all tables in the database. They can look at everything, but change nothing. An example of a Data Role name for the production environment at database-level for select authority only for the accounts receivable data warehouse AcctRecvWhs is DRole_Prod_Dbase_Select_AcctRecvWhs. All tables in the data warehouse would need to have select authority granted to this Data Role. For a non-relational database, all files in the database would need appropriate file and folder permissions assigned to the Data Role. The Data Role is then assigned to the Application Roles that require it and the Application Roles are assigned to the necessary Business Roles.

Schema-level security

Schema-level security gives authority to a group of tables within a database, but not all tables. This is useful for large, enterprise-wide databases where multiple applications have tables within the same database. Joining tables together across databases has overhead which can be avoided by having all of the tables in one database. For example, a company could have all of its tables for all of its financial applications like purchasing, accounts payable and accounts receivable in the same database with different schemas for each application. In this example, the database name could be FinanceDb and the application schema names Purch, AcctPay and AcctRecv. Secure user IDs could be assigned to Data Roles DRole_Prod_Dbase_All_FinanceDb, DRole_Prod_Schema_All_Purch, DRole_Prod_Schema_All_AcctPay and/or DRole_Prod_Schema_All_AcctRecv to receive authority for the tables they need. Users who need select authority for doing ad-hoc reporting could be put into any of the Data Roles DRole_Prod_Dbase_Select_FinanceDb, DRole_Prod_Schema_Select_Purch, DRole_Prod_Schema_Select_AcctPay and/or DRole_Prod_Schema_Select_AcctRecv, depending on their needs. Database administration is often decentralized for each application, too, so database administration authority can be granted to any of the DRole_Prod_Schema_DbAdm_Purch, DRole_Prod_Schema_DbAdm_AcctPay and/or

DRole_Prod_Schema_DbAdm_AcctRecv Data Roles. Another useful way of using schemas is to segregate security-related tables or tables with confidential information into their own schemas. For a non-relational database, schema-level security can be used for any logical grouping of tables desired.

Table-level security

Table-level security has more overhead than database- or schema-level security because more Data Roles are involved. Therefore this should be restricted to an as-needed basis. This is typically used when a user needs a higher level of access to one or more tables. For example, a user may need full authority to a personal cross-reference table used to aggregate totals from other tables for consolidated reporting or data mining. A good example of this is someone who wants to compare sales or other information across a group of similar stores. They can insert rows into a store control table that defines all stores they want to compare and then join that table to another table that contains the information they want to compare for all stores. Another example is for grouping similar items into a category or classification and summarizing sales information to the category or classification level. The user may need full authority to this control table for setting up the appropriate items within each category or classification. Databases can contain hundreds or thousands of tables, so for operating systems with limited group name lengths, you may need to assign a sequential number or alphanumeric key to each table and use that value for the table name in the data security group name.

Granting Authority to the Data Roles

In a typical environment, for each database there will be two database-level Data Roles. One Data Role will give inquiry access and the other will give full authority for all tables in the database. Within the database, GRANT commands must be executed to grant authority for the individual tables to the appropriate Data Roles. These grants can be done as part of the initial creation of each table. Once the Data Roles have been created and appropriate authority granted to them, security administrators assign Application Roles to the appropriate Data Roles to grant or revoke database access. For example, an Application Role for ad-hoc reporting could be added as a member of the database- or schema-level Data Role for inquiry access, depending on their needs.

Segregation of Duties

Business Roles are assigned Application Roles based on segregation of duties issues. Limiting users to having one and only one Business Role means segregation of duties issues can *only* arise when new Business Roles are created or existing ones are modified. Once business roles and their associated application roles have been defined and audited for segregation of duties issues, they rarely change. When they do change, the changes are usually not very significant. For this reason, after the initial segregation of duties review has been done, a complete review of all roles and their privileges should only need to be done annually. This is because each time the application roles for a business role are changed or a new business role is created, only that one business role needs to be reviewed for segregation of duties issues. This effort is negligible compared to reviewing all business roles and their privileges. It also involves much less effort than reviewing all users who are assigned that business role. This architecture for role-based security provides secure control of segregation of duties, allows users to change Business Roles whenever necessary without concern about segregation of duties issues and minimizes the effort involved in segregation of duties reviews.

Conclusion

Hopefully this helps clears up confusion about role-based security. Keep in mind this is only one architecture. Others are definitely possible and may work better, depending on the situation. However, this architecture has the benefits of being very logical and simple. That's usually a pretty good sign.