

Defense In-Depth to Achieve “Unbreakable” Database Security

Qiang Lin, Ph.D

Abstract—Enterprises realize that sole reliance on generic security mechanisms does not provide the protection they need for their business operations. More than just their employees need access to data in the database. Their partners and customers have access to these data as well. Therefore, their database cannot be hidden behind a firewall. However, no single security solution can properly protect a database. What is most important is the concept of “defense in-depth” to achieve “unbreakable” database security. This article introduces the Multi-layer Database Access Control Architecture (MDACA) based on available access control mechanisms from four major commercial databases – IBM, Microsoft, Oracle and Sybase to address the database security based on the concept of “defense in-depth”.

Index Terms—Access Control, Database, and Defense in-depth.

I. INTRODUCTION

Security conscious enterprises realize that sole reliance on generic security mechanisms, such as firewalls, intrusion detection, operating system hardening, and virus protection, does not provide the protection they need for their business operations [1]. These mechanisms are important, but insufficient to address an enterprise’s overall information security needs. Generally, the heart of today’s enterprise information system is the database, which serves as the warehouses of digital information and holds enterprises’ most critical assets. More than just their employees need access to data in the database. It is imperative that partners and customers have access to these data as well. This means that their database cannot simply be hidden behind a firewall. As the database becomes more exposed to Internet, it must properly secure it from threats and vulnerabilities of the outside world.

However, no single security solution can properly protect a database. What is most important is the concept of “defense in-depth” to achieve the “unbreakable” database security. This means that more than a single layer of security is required to adequately protect a database. A good example of “defense in-depth” is a castle. The castle contains multiple defense systems – a moat, castle walls, archers on the walls, etc. Individually, each defense system would not be able to deter an

attacker, but combined the castle becomes very difficult to penetrate [2]. Therefore, it is necessary to design a multi-layer database access control solution. The remaining of this article introduces the Multi-layer Database Access Control Architecture (MDACA) based on the available access control mechanisms from four major commercial databases – IBM DB2 V8, Microsoft SQL Server 2000, Oracle9i and Sybase ASE 12.

II. MULTI-LAYER DATABASE ACCESS CONTROL ARCHITECTURE

Fig. 1 illustrates the MDACA, which is composed of the database user accounts, user information, LDAP-based directory, application context, privileges, roles, policies, stored procedures, and data encryption.

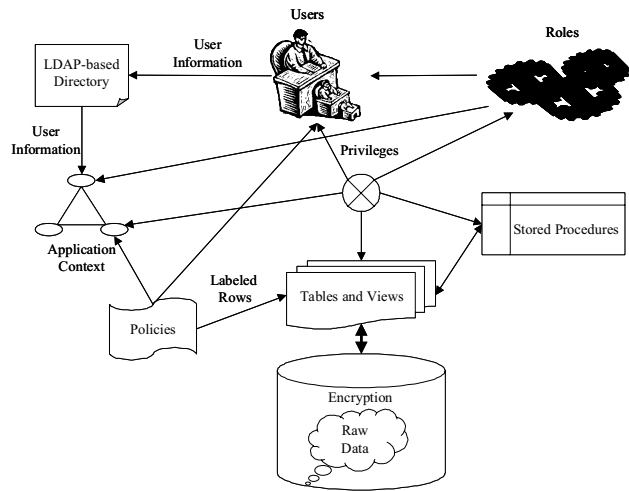


Fig. 1 Multi-layer Database Access Control Architecture

A. User Account and Privilege

Generally, before a user begins to access a database, a new database account is created for the user in the database. Then, a set of privileges will be granted to the user’s account based on the user’s needs. The typical privileges are:

- CONNECT to the selected instance of the database
- SELECT, INSERT, UPDATE and DELETE on the tables and views
- EXECUTE on the stored procedures
- CREATE and DROP to create and drop tables, views, indexes, etc.

These privileges are granted to the user by either the

Q. Lin is with Science Application International Corporation, 5180 Parkstone Drive, Chantilly, Virginia USA 20151 (telephone: 703-227-8087, e-mail: qiang.lin@saic.com).

database object owners or database administrator. In Microsoft SQL Server 2000 and Sybase ASE 12, a database privilege is known as the permission.

A user must pass through two stages of security when working in a database: authentication and authorization. The authentication identifies the user using the login information and verifies only the ability to connect to the instance of the database. If authentication is successful, the user connects to the instance of the database. Then, the user needs privileges to access the database objects. Without appropriate privileges, the authenticated users could not do anything towards the database. For example, if the user has been granted with SELECT privilege on Employee table, the user can run SELECT statement against Employee table. However, the user cannot insert a new record into the table nor delete the records from it.

Database user accounts and privileges are considered as the first layer of the MDACA. After the user account being created and granted with a set of privileges, the user can work on the database by any means. For example, the user can work on a Microsoft SQL Server 2000 database using SQL Analyzer, ISQL, Access, or any Ad hoc reporting tools. It is not possible to limit which tool a user allows to accessing the database.

B. Role

A database role groups multiple privileges as a set so that they can be granted to and revoked from users simultaneously. A user account can be a member of any number of roles within the same database and can hold privileges appropriate to each role. A role can also be granted to the other roles. Therefore, roles can simplify security administration in databases with a large number of users or with a complex security system [3]. For example, one can grant the role, rather than many individual privileges, to those users who run the application. Then, as employees change jobs, you need to grant or revoke only one role, rather than many privileges. One can change the privileges associated with an application by modifying only the privileges granted to the role, rather than the privileges held by all users of the application [4]. In IBM DB2 V8, a database role is known as the database group.

A database always provides some predefined roles to help in database administration. These roles are automatically defined for the database during the database creation. One can grant privileges and roles to, and revoke privileges and roles from, these predefined roles in the same way as the other user or application defined roles.

To simplify application privilege management, one can create a role for each application and grant that role all the privileges a user needs to run the application. In fact, an application might have a number of roles, each granted a specific subset of privileges that allow greater or lesser capabilities while running the application. Oracle9i also allows roles to be managed centrally. One can create roles such that their use is authorized using information from the operating

system, a network authentication service, or an LDAP-based directory. Central management of roles provides many benefits. If an employee leaves, for example, all of the employee's roles and privileges can be changed in a single place [4].

In general, a role is enabled automatically after its creation. However, a role in Oracle9i database must be enabled for a user before it can be used by the user. As mentioned previously, privileges granted directly to a user are always available to the user; therefore, directly granted privileges cannot be selectively enabled and disabled, depending on the user's current task. By contrast, privileges granted to a role can be selectively made available to any user granted the role in Oracle9i. The Oracle9i roles can be default so that they are automatically enabled for a user when the user creates a session. If a role is not granted as default for the user, the role can be enabled through a SET ROLE statement. It is recommended by Oracle that:

- Enable the proper role when the application starts, and
- Disable it when the application terminates.

Combined with different role authorization mechanisms, the database role can be treated as another layer of the MDACA.

C. Stored Procedure

In Microsoft SQL Server 2000 and Sybase ASE 12, a stored procedure is a group of Transact-SQL statements compiled into a single execution plan [5]. In Oracle9i, a stored procedure can be written either in PL/SQL or in Java. PL/SQL is a block-structured language. That is, its basic units (procedures, functions, and anonymous blocks) that make up a PL/SQL program are logical blocks, which can contain any number of nested sub-blocks. Typically, each logical block corresponds to a problem or sub-problem to be solved [6]. In IBM DB2 V8, a stored procedure can be written in C, PL/I or Java.

Stored procedures assist in achieving a consistent implementation of logic across applications. The SQL statements and logic needed to perform a commonly performed task can be designed, coded, and tested once in a stored procedure. Each application needing to perform that task can then simply execute the stored procedure. Coding business logic into a single stored procedure also offers a single point of control for ensuring that business rules are correctly enforced. Stored procedures can also improve performance. Many tasks are implemented as a series of SQL statements. Conditional logic applied to the results of the first SQL statements determines which subsequent SQL statements are executed. If these SQL statements and conditional logic are written into a stored procedure, they become part of a single execution plan on the server. The results do not have to be returned to the client to have the conditional logic applied; all of the work is done on the server [5].

As another way to restrict users from exercising application privileges by way of different tools is to encapsulate privileges into stored procedures. Grant users execute privileges on the procedures, rather than issuing them direct privilege grants. In

this way, the logic goes with the privilege. This allows users to exercise privileges only in the context of well-formed business applications. For example, consider authorizing users to update a table only by executing a stored procedure, rather than by updating the table directly. By doing this, you avoid the problem of the user having the `SELECT` and `UPDATE` privileges and using them outside the application [4]. Therefore, stored procedures form one more layer of the MDACA.

D. Policy and Row Level Access Control

The row level access control (also known as fine-grained access control in Oracle9i) allows one to build applications that enforce security policies at a row level of granularity. One can use it, for example, to restrict a customer who is accessing a database to see only the customer's own account, a physician to see only the records of the physician's own patients, or a manager to see only the records of employees who work for the manager. To use the row level access control, it needs to create security policies attached to the database tables and views for the given application. Then, when a user enters a `SELECT` or `UPDATE` statement on the table or view, the database dynamically modifies the user's statement, transparently to the user, so that the statement implements the correct access control [4]. Currently, only Oracle9i and Sybase ASE 12 support the row level access control.

The major component of the row level access control is access rules or policy. These rules can define any security policy expressible in a `WHERE` clause, without a developer having to hard code the policy in an application. The access rules are, however, compiled as `WHERE` clauses within a query so the additional protection does not affect performance. In addition, access rules can query Java-enabled external databases and LDAP directories, and use information stored there as part of the access control decision-making process. As an example, consider a security policy for an Application Service Provider (ASP) stating that each external customer is only permitted to access information relating to his own company. To enforce this security policy, the ASP defines an access rule on the company name column, `(C) = "company_name,"` in table T. When a user submits the query `"select * from T,"` the access rule activates and enforces the policy that the user is only able to see rows of information pertaining to his own company [1].

Oracle9i has added the Oracle Label Security. With this option Oracle9i can enforce row level access control and label-based access control automatically. For example, you can label data Company Confidential or Partner Releaseable to automatically limit access to data based on the label of the data and the labels of data a user is permitted to access. By using Oracle Label Security, organizations can implement row level and label-based access control quickly, in many cases without additional programming [4].

With the row level security, although a user granted with

certain privileges on a table, can only operate on the limited rows in the table defined by the policy. Therefore, it becomes an important layer of the MDACA.

E. Application Context

For many commercial packaged applications, application users are not database users, especially for many Web-based applications. For these applications, multiple users authenticate themselves to the application, and the application then connects to the database as a single, highly privileged user. Oracle names this the "One Big Application User" model [4]. These applications generally cannot use many of the intrinsic security features of the database, because the user identity is unknown to the database. Thus, several layers of the MDACA introduced in before, such as user privileges and roles, cannot be used.

Applications that use the One Big Application User model must build security enforcement into the application rather than using database security mechanisms. In this case, since it is the application, not the database, to recognize users, the application must enforce any per-user security measures itself. This means that each application to access the database must implement its own security [4]. For example, if an organization implements a new human resource application, it must also implement security to ensure that users do not get more access through the other tools than they would have in the application itself. Security becomes expensive because organizations may have to implement the same security policies in more than one application, which is also error-prone. Even worse, if the single, highly privileged user account is compromised, the database that drives the application could be severely damaged or even destroyed. Therefore, Oracle recommends that, where possible, the applications should be built that the application users are the database users. In this way, the application can leverage the intrinsic security mechanisms of the database. When security is enforced in the database itself, rather than in the application, it cannot be bypassed.

For an organization, it may not be difficult to synchronize up to hundreds of users among applications and databases. However, it becomes the nightmare for the enterprise administrators to synchronize thousands or more users among many applications and multiple databases. Therefore, many new enterprise applications begin to implement the single login security, in which organizations centralize user information and management in an LDAP-based directory. To still enforce different layers of the MDACA on the database without creating actual database user accounts, both Oracle9i and Sybase ASE 12 have introduced application context. Each application can have its own context with its own attributes. For example, one organization has three applications: General Ledger, Order Entry, and Human Resources. The organization can specify different attributes for each application. Thus,

- For the General Ledger application context, you can specify the attributes `SET_OF_BOOKS` and `TITLE`.

- For the Order Entry application context, you can specify the attribute `CUSTOMER_NUMBER`.
- For the Human Resources application context, you can specify the attributes `ORGANIZATION_ID`, `POSITION`, and `COUNTRY`.

In each case, the organization can adopt the application context to its precise security needs. Application context attributes can be stored in an LDAP-based directory and assigned to one or more enterprise users. They can be retrieved automatically upon login for an enterprise user, and used to initialize an application context.

Application context is especially helpful if the security policy is based upon multiple security attributes. For example, a policy function which bases a predicate on four attributes, such as employee number, cost center, position, spending limit, would have to execute multiple sub-queries to retrieve this information. If all of this data is already available through application context, then performance will be much faster [4]. In Oracle9i, using application context with row level access control is called virtual private database. Therefore, the properly designed application context combined with row level access control formulates a powerful and effective layer of the MDACA, especially useful for the enterprise applications.

F. Database encryption

Once all the access controls described in previous sections are in place, encryption should then be implemented as the innermost layer of the MDACA. Encryption provides an additional restriction if access controls are circumvented. In other words, encryption should stop someone who has already broken through multiple layers of the MDACA.

Another benefit of encryption is to limit the administrative power. The administrative accounts begin as anonymous and all-powerful, thus enabling administrators to access any information they want without being held accountable. Enterprises need to focus on constraining administrative privileges and on holding individual administrators accountable for their actions [1]. Although the row level security can restrict the administrator to access data in the database, a malicious administrator still can change the data owner's password and login as the owner. Encryption prevents the administrator to view the encrypted data if only the data owner holds the encryption and decryption keys.

While there are many good reasons to encrypt data, there are many bad reasons to encrypt data. Encryption does not solve all security problems, and may even make some problems worse [4]. Two points to understand about encryption are the following:

- Encryption does not solve access control problems – they are responsible by the other layers of the MDACA.
- Encrypt everything does not make data secure – encrypted data can be deleted or modified.

It is important to maintain the proper backups so that if someone deletes or changes the encrypted data, they can be

restored [2].

1) Encryption of "data-in-motion"

Encryption can be categorized into two types – encryption of "data-at-rest" and encryption of "data-in-motion". The issues involved with providing each type of encryption are very different.

Encryption of "data-in-motion" hides information as it moves across the network from the database to the client or from the client to the database. "Data-in-motion" includes traffic moving over your local network, the Internet, or even over a wireless network [2].

For example, both SSL (Secure Sockets Layer) and Kerberos network encryption are integrated into Sybase ASE 12. In addition to protecting the confidentiality of sensitive information in transit, each solution authenticates an ASE server's identity. SSL uses public key cryptography and authenticates a database's identity through a digital certificate, while Kerberos mutually authenticates a database and a user when the user logs on [1].

2) Encryption of "data-at-rest"

Encryption of "data-at-rest" involves transforming information stored in the database into a form so that it is only readable by authorized individuals. Both Oracle9i and Sybase ASE 12 support encryption of "data-at-rest".

Sybase ASE 12 utilizes Protegrity's Secure.Data to encrypt information stored in it. Since encryption operations are incorporated into Sybase ASE 12 and are an integral part of query processing, neither application developers nor end-users have to worry about managing and protecting encryption keys or performing encryption operations. Secure.Data administrators handle all of the key management operations. With Secure.Data solution, individual columns can be selectively encrypted and a different encryption key may be used for each column within a single database or even across different databases [1].

Oracle9i provides a PL/SQL package to encrypt and decrypt stored data. The package, `DBMS_OBFUSCATION_TOOLKIT`, is provided in both Standard Edition and Enterprise Edition Oracle9i. This package supports bulk data encryption using the Data Encryption Standard (DES) algorithm. Key management is programmatic. That is, the application (or caller of the function) must supply the encryption key; and this means that the application developer must find a way of storing and retrieving keys securely [4].

Special difficulties arise in handling encrypted data that is indexed. If the data in a column are encrypted, then an index on that column will also contain encrypted values. Then, the index is essentially unusable for any other purpose, such as equality checking [4].

III. SUMMARY

This article has defined the MDACA to implement the concept of "defense in-depth" to achieve the "unbreakable" database security. Note that the MDACA is based on the

available access control mechanisms from four major commercial databases – IBM DB2 V8, Microsoft SQL Server 2000, Oracle9i and Sybase ASE 12. Apparently, both Oracle9i and Sybase ASE 12 have all the mechanisms required for the MDACA. However, each application and organization may tailor the MDACA for implementation to suit the actual needs.

REFERENCES

- [1] "Information Security: Beyond the Firewall," *White Paper*, Sybase, 2002, <http://www.sybase.com/content/1022216/5335securitywpapv4.pdf>.
- [2] "Encryption of Data at Rest – Database Encryption," *White Paper*, Application Security, Inc., <http://www.appsecinc.com/techdocs/whitepapers.html>.
- [3] "Administering SQL Server – Managing Security," *MSDN*, Microsoft, <http://msdn.microsoft.com/library>.
- [4] "Oracle9i Application Developer's Guide – Fundamentals, Release 2 (9.2)," *Part Number A96590-01*, Oracle, http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96590/adgsec01.htm#1004586.
- [5] "SQL Server Architecture – SQL Stored Procedures," *MSDN*, Microsoft, <http://msdn.microsoft.com/library>.
- [6] "PL/SQL User's Guide and Reference, Release 2 (9.2)," *Part Number A96624-01*, Oracle, http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96590/adgsec01.htm#1004586.