Global Information Assurance Certification Paper

Developing A Secure Oracle Database Application

By William T. Abrams
GSEC v1.2f
December 27, 2001

Abstract

The goal of information security is to protect your vital information resources and reducing – or eliminating – the threat to its confidentiality, integrity, and availability. For an Oracle database application, this requires a concentration of security efforts within your Oracle "environment" i.e. the application, the Oracle run-time environment, and the database itself.

This paper will focus on some of the vulnerabilities found in a standard Unix Oracle installation, the application of Oracle security measures within a Unix-based Oracle environment, and within the database itself, and specifically on the new "fine-grain" security access capability of Oracle 8i.

Introduction

It would seem that most information security-related articles and headlines address the more glamorous, sophisticated technologies such as firewalls, packet sniffers, and intrusion detections devices. However, since the primary goal of security is to protect information and access to it, I would think it more appropriate and cost-beneficial to focus more attention and resources to the application of security features at the database level and the systems immediately surrounding those applications. Fortunately, there is a growing number of security professionals adopting this perspective:

> "When people talk about security today, they tend to focus on the edge of the network, where they deploy firewalls and VPN software to secure access to the network. The trouble is, this gives IT people the illusion that the entire enterprise is secure when they have really just set up a first line of defense. Once you secure the network, the second line of defense is the applications themselves…Much greater attention should be paid to making the applications secure from intruders. Failing that, a third tier of defense should be set up around the data in the applications" {reference [1] }

I emphasize cost beneficial, since by securing the "back end", you really get twice the payback -- securing your databases and applications that access them reduces the security threat irrespective of whether the threat vector is ***internal*** - from employees, contractors, and others who already have access to internal portions of the company network and information resources- or ***external*** –i.e. from those non-authorized users or would be attackers.

As for databases, the Oracle Relational Database Management System (Oracle RDBMS) has certainly received its fair share of attention from the security perspective, undoubtedly due as much from its dominant position in the marketplace, as from its reported security shortcomings.

Fortunately, each successive Oracle software version has reduced the number of security vulnerabilities, and work-arounds exist for most earlier problems.

In the first part of this paper I will address a number of these vulnerabilities in the Oracle software environment itself, as reported in references [2] through [5].

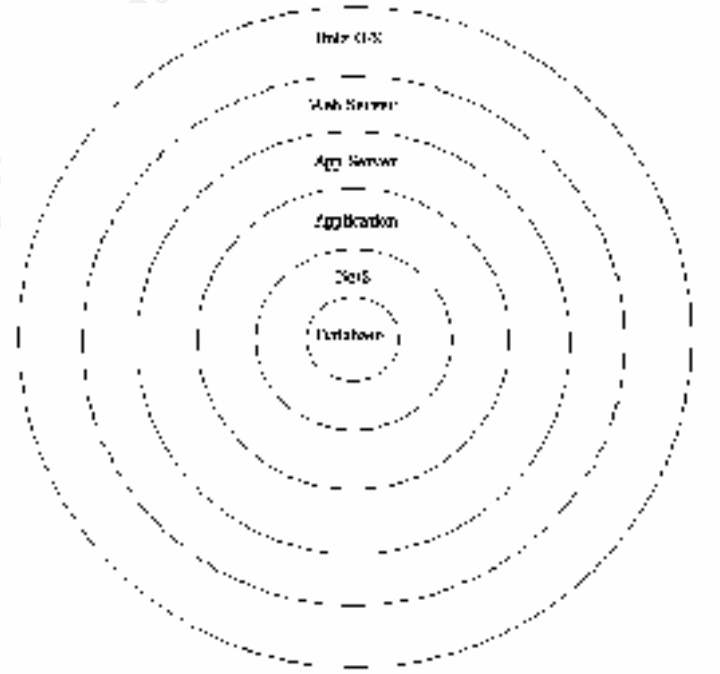
From the database perspective, there are certain features of the database creation process itself, as well as certain capabilities provided for administering the database, which can also become a source of security problems, as reported in reference [6]. These I will also discuss, as well as provide some guidance.

Fortunately, the Oracle RDBMS is also rich in features and utilities which allow applications developers to build in security from the ground up. A few of these features were identified in reference [7]. In the final section of this paper, I will provide some examples of these, and more specifically provide some focus on one important feature not discussed in [7]: that is, the Oracle 8i row-level, or "fine-grain" access capability, as described in references [8] through [11].

Defining and Securing the Oracle Environment

The development and operational environment for the Unix-based database application is comprised of many levels, or layers, of software. Actions must be taken at each successive level to enhance the security of underlying applications and data. The more difficulty each successive layer poses to an attacker or unauthorized user, the more likely he is to either quit, or fail.

As depicted here, the real "treasure" we are trying to secure is the data itself. It is at this core level, in the database, that the most stringent security measures should be taken. Analogous to the security applied to safeguarding of economic treasures in a bank, the vault itself has the strongest, most difficult to penetrate. This is the ultimate barrier to the attacker, and poses the most difficulty to the attacker even after he has penetrated the outer layers of security.



First, let's define the scope of the Oracle environment which we will address. As seen in the figure above, this paper will focus specifically on the 3 innermost levels of the Oracle environment: the RDBMS "engine" and associated software components, Oracle applications such as SQL*Plus, Net8 (formerly SQL*Net), and the database proper.

The Oracle RDBMS and associated software can, depending on the package(s) purchased, consist of a fairly long list of software bundles. It's beyond the scope of this paper to identify all of them, nor even to give a detailed and complete description of any. However, the more major pieces include:

1 RDBMS – which provides the programs for running the entire database management system and the associated processes
2 Administrative utilities, such as
    - Server Manager (for starting up, shutting down, and other administrative activities)
    - SQL*Loader (for loading bulk data e.g. from an ascii file)
    - Import/Export (for exporting the data from a given database, schema, or table; into an Oracle-readable format for possible later "import" into a new or empty database…etc)
3 SQL*Plus –which provides an environment for directly interacting with the database, executing SQL commands, and the like.
4 PL/SQL –which provides a Procedural Language capability for programming of procedures and triggers
5 Net8 (formerly SQL*Net) –which provide the communications network means for client processes to communicate and interact with the database.

The initial step taken prior to loading the Oracle software is the creation of the Unix user to own the Oracle software. Unfortunately, the typical approach to this is to name that user "oracle". Though this may simplify a few aspects of system administration, it is also so frequent that any potentail attacker already has useful information. The initial security step one can take is to create some other name for the user who owns the Oracle software.

The owner of the Oracle software is also made a member of the unix group *dba*, and is afforded to umask value of 022. In a Unix system, this essentially sets default settings for files created by that owner to be defaulted to permissions of 644 for normal files, and 755 for executables. For those unfamiliar with Unix, the setting of 644 grants read and write permission to the file owner, and read-only permission to members other members of *dba* group, and to *others* ("world*")*. The 755 setting will additionally also grant permission to *execute* the file to the owner, members of *dba* group, and world.

It has been pointed out that in the event files or directories were pre-created, i.e. prior to loading of the Oracle software, that the newly loaded files overwrite the prior-existing files, ***but retain the original permissions*** {reference [5]). To be on the safe side, the system administrator should ensure, therefore, that the loading of Oracle software are initially pointed toward non-existing directories. And after loading the software, go back through the installation and ensure appropriate permissions on directories and files.

It has likewise been determined that, in early versions of Oracle 8 and prior version 7, many of the executable files were created with the SetUID-bit set. These vulnerabilities can be found from many sources on the internet, including references [2], [3], [4], and [5].

Some of the executables are *otrccol, otrccref, otrcfmt otrcrep* (used by Oracle Trace utility), *oratclsh* (used by TCL debugger), *dbsnmp* (also called 'Oracle Intelligent Agent'), and the *oracle* binary itself. Because of the Setuid permission, the binary runs with the elevated privileges of the owner, and can be used to exploit certain buffer overflow conditions that exist.

The basic work-around (pending application of patches or upgrades to latest Oracle version), consists primarly of re-setting permissions for these executables, for the "trace" utility executables.

% chmod -s otrccol otrccref otrcfmt otrcrep
% chmod 751 otrccol otrccref otrcfmt otrcrep
-- reference [2a]

For the oracle executable itself, the execute privilege should be taken away form the rest of the "world " – (Others)

% chmod o-x oracle -- reference [2b]

However, there is an additional *caveat* here -- reference [2b]:

> "The workaround suggested above will permit only the owner of the oracle executable and users defined in the OS DBA group to run the oracle executable directly. With the execute permissions for "others" removed, other users cannot connect to an Oracle database server using the BEQ driver. … To avoid this problem, local users must connect to an Oracle database using the IPC driver that makes it possible to connect to a TNS listener listening on an Oracle "

It needs to be emphasized that the above list is not a complete list of Oracle vulnerabilities. References [2] through [5], and other similar sites, should be queried extensively by Oracle system and database administrators, for applicable Oracle vulnerabilities for their specific products ands versions.

One other notable vulnerability lies with the Oracle listener control utility. This utility is used to startup, shutdown, and otherwise set configuration parameters for the Net8 (formerly SQL*Net) listenr processes. These are the processes which listen for attempted communications with the Oracle database from clients, and set up communication sessions on behalf of those clients. The principal problem is that the program for controlling the listener, ***lsnrctl*** , does not require a password ! Rather, as long as Oracle environmental variables are set correctly (various paths, Oracle home, oracle instance, …) a user of "other" group setting can invoke the utility, and at the very least start gathering important information.
Note, however, that the ***lsnrctl*** program does allow for a password to be set. Unfortunately, there is neither an enforced expiration, and the password itself is stored insecurely. (reference [5]).

Obviously, administrators should at the very least set a strong password for lsnrctl, and then pursue some of these other issues further.

Securing the Database

Normally a database is created as part of, or shortly after, the Oracle RDBMS software loading (and executable generation) phases. In this section I will point out relevant security features provided by the Oracle database structures, and several security issues which arise immediately after the initial database creation.

In a relational database, data is viewed as consisting of individual records - called rows - stored in a basic structure, or -object - called a *table*. The table is seen as 2-dimensional. One dimension is referred to as "columns" (which define data attributes, or fields), and "rows", which represent a single record of data with a value (or null) for each column.

Other basic objects, besides tables, include:

- *Indexes* – used to increase performance in accessing or updating rows; also to enforce certain constraints on the data
- *Views* – used to restrict the columns of a table allowed to be viewed by some users or groups of users. Also used to simplify a more complex joining of data from 2 or more tables. {These are discussed in more detail later}.
- *Procedures* – used to perform a routine or set of actions on data; explicitly invoked by the user, a procedure, or trigger
- *Triggers* – also used to perform a routine or set of actions on data, but "triggered" or initiated by some event which has been defined, such as after updating a field (column), after deleting a record, and so on.

Triggers can be used to perform a wide variety of tasks, including actions which relate to security. They are commonly used to implement database auditing tasks. For example, a trigger can copy a row of data into a separate, restricted table prior to update or deletion, along with the identity of the user and a timestamp performing the action. This would preserve an audit trail of changes to that data record. Also it should be noted that the trigger executes within the privileges of the table owner, and not at the "whim" of the user taking action on that table. The user cannot force a trigger to react or fail to react.

Associated with the database, and with these objects defined above, are *privileges*. Privileges are defined at 2 levels: the "system" level, and the "object" level.

System privileges permit the execution of database activities related to the entire database operation itself, and to activities allowing creation/deletion of database objects. There are over 100 system privileges, a few of which include:

- **create tablespace** - which allows for file creation, and hence could impact overall platform operation, and disk storage,
- **create *object*** - where *object* could be any of those identified previously,

- **select any table** – which would allow a user to select data from ANY table in the database.
- **alter database** – which allows for setting of certain database configuration parameters
- **create session** – which enables a user to connect to the database, as himself, and start a session.

For obvious security reasons, these system privileges must obviously be very tightly controlled, and administrators would do well to follow the "principle of least privilege" credo. From a security standpoint, normally users need only create session system privilege, since they would not normally create their own objects.

Object privileges are those privileges associated with the manipulation of actual data within the objects. There are about eight object privileges , the principle ones being :

- **select** – the ability to read data from a table (or view)
- **insert** – the ability to create new rows of data
- **update** – the ability to update column values of existing data rows
- **delete** – the ability to delete rows from a table
- **execute** – the ability to execute a procedure (or set of procedures called a package)

Again, the credo of the "principle of least privilege" is an administrator's best bet for increasing the security posture of his database application. A further point is made in Reference [6], relative to the inherent capability to pass on privilege granting control :

> "Another security hole is Oracle's ability to grant control authority to another user. Anyone with control authority can give these super user rights to another, which can cause serious problems. For example, Bob, the CFO, gives Mary control authority to the Salary table of the Accounting database. Mary … grants control authority to Susie. Cindy, Janet, and Jill pressure Susie… into granting them control access to the database. When Mary returns from maternity leave, Susie's authority is revoked; however, this does not affect Cindy, Janet, and Jill's control authority. This revocation of access rights does not cascade down to their level."

The real problem with this feature is that it can not be restricted by the database administrator. Any user (or schema) can grant this "with grant option" on his own objects to other users. It therefore is incumbent on the schema owner to ensure this privilege is stringently controlled.

Roles are helpful administrative tools, allowing privileges to be grouped together and granted to users (or even other roles) as a set of privileges. This greatly increases the manageability of security administration of a multiple user application.

A *user* is essentially an entity defined by the database administrator (dba) , and hence known to the database system, who can connect to the database for a variety of purposes:

- owning data (a special type of user called a *schema*)
- accessing data (your typical application user)
- performing database administration duties

just to name a few. Upon attempted login, or connection to, the database, users are authenticated through the use of an assigned password, which is stored in encrypted form in the database.

One very important security aspect for database application design is that each user must have his own personal id and password. It is unfortunate that so many applications are developed which require multiple users to access through the use of a common username/password. This is a terrible idea from a security standpoint.

> "…when a userid/password is a community resource, then the database administrator has surrendered all pretext of security. Every time a user leaves the department or the company the keys to the treasure vault go with him…" (reference [8]).

Secondarily, following an actual or suspected security breach, any benefit of database auditing has likewise gone out the window ! This practice also then allows for the frequent change of the schema's password to reduce the likelihood of someone gaining direct access to the schema with higher levels of data privilege.

After initial database creation, there are a few issues to be addressed by the administrator from a security perspective. As pointed out in references [5] and [6], there are a large number of default users (schemas) created, with default passwords. A large number of these may never be referenced again by the database community, and should therefore be dropped. But at the very least, the passwords should be changed immediately.

Reference [5] also points out that by default certain data dictionary views such as ALL_USERS (which contains names of all database users, as well as other information) are accessible to the *Public* – meaning everyone. It would therefore be strongly advisable for security administrators to closely examine the need for public access to these "ALL_*xxxx*" views, and restrict again according to the principle of least privilege.

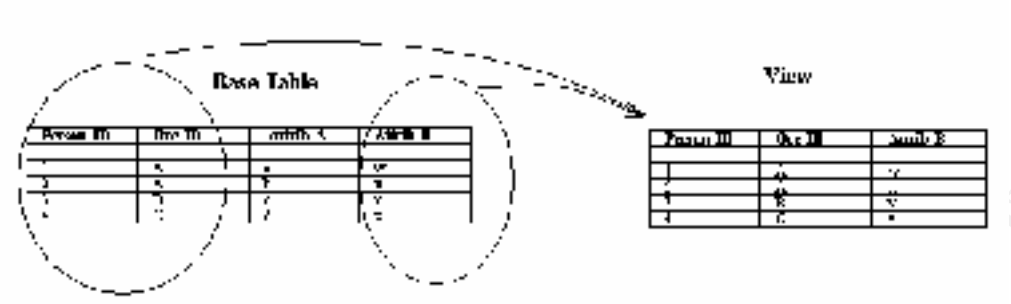
Securing the Data

To illustrate the implementation of security within the database design itself, we'll use a simple base table, comprising rows of data (commonly called records), with several attributes for each row – called "columns".

Base Table

Person ID	Org ID	Attrib A	Attrib B
1	A	a	w
2	A	b	x
3	B	c	y
4	C	d	z

Since some of these columns may represent data which is confidential – and should be restricted from one or more groups of users – a more restricted "view" of this table is needed. Hence a *view* is created, wherein the restricted data, e.g. Attribute 'A', is not included:



This view may then be the object which is accessed from within a given application; users then are unable to access the 'attribute A' values. These values could represent any data requiring special access restrictions, such as financial data, credit card information, personal information, etc. Note that a view can be hardcoded to return only specific rows, but accessing this view would then return the same rows to any user granted select privilege. Also, a large number of views would have to be created to satisfy all possible queries !

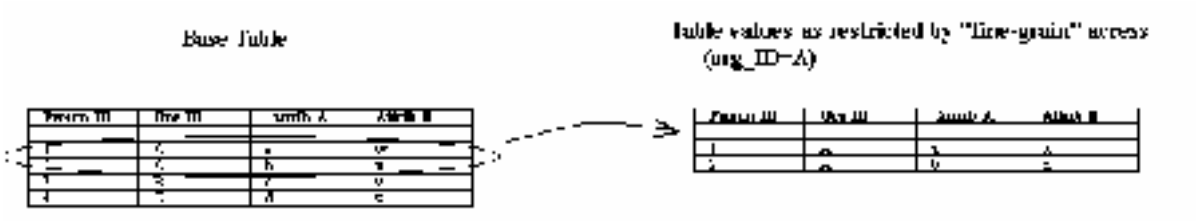
Fine-Grain (Row-Level) Access

Although views certainly are useful for securing the database application, this single feature can obviously only provide a single dimension of security. What about row-level security, where only specific rows should be available for view by a single person, or group of users ?

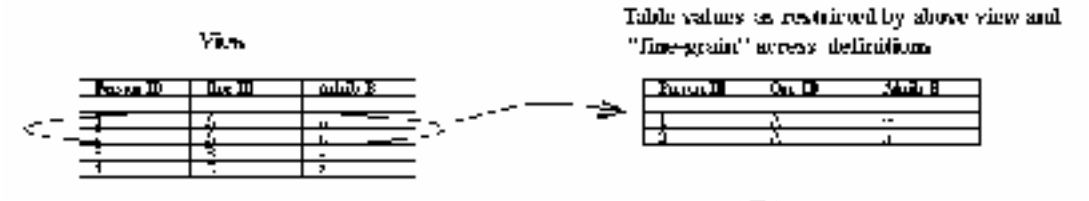
This has always been a bit trickier to apply, as the system developer and programmer needed to develop and implement more complex coding, such as use of more complex triggers, to implement at the row level. Additionally, these may have had to be further applied to multiple tables. As described in reference [9]:

> "Prior to Oracle8i…When organizations needed to enforce finer-grained access control for data within tables, they had to rely on database views, or program the access logic into stored procedures or applications. There are several limitations with these approaches. Among them is that users who can gain direct SQL access to tables can bypass security constraints imposed by applications, stored procedures or views. Another is that programming access logic into views or applications is often cumbersome and difficult to manage…"

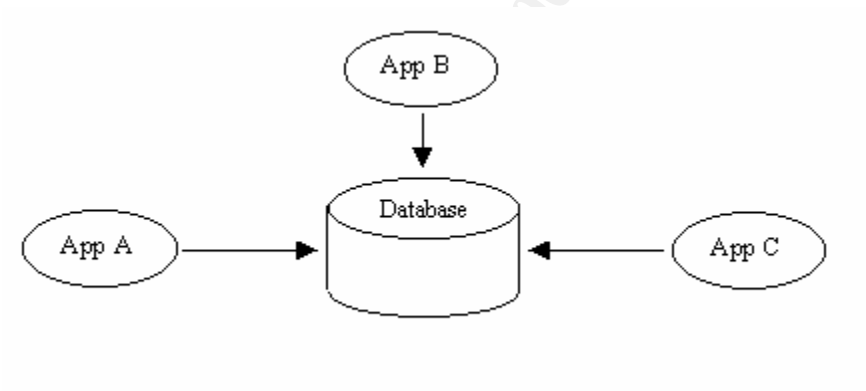
This introduction of row-level security, called "fine-grain access", within the Oracle database, is clearly of immense value to increasing the security profile of the database application. For example, using the same base table from before, implementation of fine-grain access can restrict the rows of data available to a user (or group of users) to only those of a given Organization – contained in OrgID column.



Obviously, combining this level of access, along with that of the pre-defined view, allows for even further restrictions on the data accessible.



One very real benefit of this fine-grain access control is the fact that it is implemented **within the database**, and hence imposes security at this level: the data itself is protected without regard to how the data is accessed. In other words, many applications may access the data, but a given, specific user will not be able to bypass security, and achieve greater access rights, by merely using a different application – UNLESS THIS FEATURE IS DESIRED! (see Virtual Private Database below)



A detailed description of fine-grain access, and additional "Virtual Private Database", can be found in references [9], [10] and [11]. As described therein, fine-grain access allows for a predicate (sometimes referred to as the "where" clause of an SQL statement) to be dynamically built by the system, and appended onto a users query of a table. This dynamic feature is automatic, and is implemented every time the table is accessed, as long as the table's "policy" is enabled. The policy is carried out by a pre-defined function, which takes the schema name and table name as input, and returns the predicate.

In the example previously used, the function would be programmed to retrieve some information (such as username, or the user's organization) and return ***where OrgID='A'***

whenever that table was queried by that user. This predicate would be concatenated onto any other criteria specified by the user.

Through the addition of another feature, called "Application Context", application-specific parameters can be introduced and specified at run-time. Hence, a different row-level access can be defined by the application itself. This would be most effective in a Data Warehouse environment, in which many applications may source their data from a single database. This results in what Oracle calls a "Virtual Private Database".
As you can see, this adds an extremely powerful capability to the Oracle security arsenal.

In conclusion, although there have been, and will probably continue to be, vulnerabilities within the Oracle environment, Oracle does have a good reputation for "closing" these issues in subsequent releases. Additionally, there are many internet sites, some referenced here, which provide a rich source of vulnerability information and work-arounds.

On the database application front, there are a great many features within the Oracle database for securing the data stored therein. With the newer fine-grain access feature, the data can be made even more secure from unauthorized access or modification, regardless of the access means.

References:

[1] **Limiting your security to a firewall could be akin to opening Pandora's box**
Michael Vizard, Infoworld,
URL: http://www.infoworld.com/articles/op/xml/01/04/09/010409opvizard.xml

[2] **Oracle Technology Network, Security Alerts, Current Alerts**
URL: http://otn.oracle.com/deploy/security/alerts.htm

[2a] URL: http://otn.oracle.com/deploy/security/pdf/otrcrep.pdf
[2b] URL: http://otn.oracle.com/deploy/security/pdf/oracle_race.pdf

[3] **SecurityFocus Vulnerabilities:**
URL: http://www.securityfocus.com/cgi-bin/vulns.pl

[4] **U.S. DOE-CIAC Bulletins: Oracle Corporation©**
Computer Incident Advisory Capability, (CIAC) ,U.S. Department of Energy
URL: http://www.ciac.org/ciac/bulletinsByType/vndr_oracle_bulletins.html

[5] **Protecting Oracle Databases**
Aaron Newman, Application Security, Inc.
URL: www.appsecinc.com/presentations/oracle_security.ppt

[6] **Database Security**, Bob Angell,
ITWorld.com, DATABASE MANAGER --- 10/22/2001
URL: http://www.itworld.com/nl/db_mgr/10222001/

[7] **An Overview of Oracle Database Security Features,**
Lorraina Hazel, CNE, May 13, 2001 (SANS GIAC Certification Paper),
URL: http://www.sans.org/infosecFAQ/appsec/oracle.htm

[8] **Essential Oracle8I Data Warehousing**,
Gary Dodge, Tim Gorman (Wiley Publishing, 2000)

[9] **Securing Three Tier Systems with Oracle 8I,**
John H.Heiman, Oracle
URL: http://otn.oracle.com/deploy/security/oracle8i/pdf/threetier.pdf

[10] **The Virtual private database in Oracle8I**,
An Oracle technical White Paper, Nov 1999
URL: http://otn.oracle.com/deploy/security/oracle8i/pdf/vpd_wp6.pdf

[11] **Creating Virtual Private Databases with Oracle8*i*,**

Mary Ann Davidson, Oracle
URL: http://www.oracle.com/oramag/oracle/99-Jul/index.html?49sec.html