How I Broke into Your Database

Kevin Loney, Kevin Loney Consulting, L.L.C.

Oracle databases are vulnerable to security threats from both technical and non-technical measures. To secure a database, you must close the most common security holes and you must secure the server and network on which the database resides. You must also secure every copy of the database, including backups and test databases. In this article, you will see the results of using different attack methods as well as measures that could have prevented each successful attack. The accompanying presentation provides further details on the social engineering methods involved.

Attack 1: Thanks For Giving Me A Copy Of Your Data

The most successful attempts to break into databases often exploit two common areas where the data is held unsecured: backups and development databases. If I can acquire a copy of your backups, then I can recreate your database on my servers, and any security reporting you have in place will be worthless. Backups can often be acquired just by asking for them. It is imperative that the personnel performing your backups understand the security issues for your databases.

For example, if the database is backed up via disk-to-disk backups, then the disk copy of the last backup must be protected against unauthorized reads. If you backup to tape, then the tape library must be secured. Past security audits have revealed enterprises that store their backup tapes in unsecured rooms or in peoples' homes.

Development and testing databases provide an even greater security threat. In many enterprises, the development database is periodically refreshed with data from the production environment. Developers are thus given full access to your production data in a usually unaudited database environment. Given that level of access, it is not surprising that most security breaches originate from within the enterprise. Databases used for testing are even more likely to be exploited, since user acceptance testing involves giving many more users access to the data in the test database. As the pool of users with access to the database increases, the potential for security breaches increases dramatically.

To prevent such security breaches, you need to control access to each copy of your production data, whether it is on a backup tape or in a development/test database. When possible, edit the data in the development/test databases so no confidential or proprietary data is revealed. Rather than using the production data for testing, you should generate test data that accurately tests the system's functionality without revealing sensitive information.

As applications are increasingly published to the Web, more databases are being declared to be 24x7 available. As a result, there are more databases using replication and disk-to-disk backup methods. The increased volume of data online presents more opportunities for unauthorized access. For example, users could attempt to access sensitive data in a replica database instead of the online production database. If you have a second database that is used primarily for support of reporting requirements, you need to make sure that database is as secure as the production database.

ATTACK 2: THANKS FOR LEAVING THE DOOR OPEN

There are well-known entrance points into an Oracle database. As new features are introduced, Oracle occasionally introduces new means of entry for someone intending to access your data without authorization.

Traditional entry points include the SYSTEM/MANAGER, SYS/CHANGE_ON_INSTALL, and SCOTT/TIGER accounts. Other entry points include:

- DBSNMP/DBSNMP. The DBSNMP account is created via the catsnmp.sql script, run as part of the catalog since Oracle7.3.3. The DBSNMP account is only used by the Intelligent Agents - if you don't use the Intelligent Agents, then you don't need the DBSNMP account. You should drop the DBSNMP account if it is not needed (and if you need it later, you can always re-create it via the catsnmp.sql script). If you use the DBSNMP account, then you should change its password. If you change the DBSNMP account password, you will need to put the new password (in clear text) in the snmp.ora file. You will need to protect the snmp.ora file against unauthorized reads.
- The oratclsh hole. As documented by several Oracle users, installing the Intelligent Agent in Oracle 8.0.5 in some operating systems sets the SUID bit for the oratclsh executable. This setting allows anyone on the system to use oratclsh to execute TCL commands as the root user. Remove the SUID bit to disable this problem. The hole exists as soon as you install the software; using the Intelligent Agent is not required.
- Application accounts. Why bother attempting to break into a user account when you can log in as the table owner? Many application accounts have far more privileges than they need. If the application is a third-party account, then there is a good chance that the password for the account that owns the application tables is seldom changed. Often, the password for that account is not changed because the application support team and the DBAs are unsure of the impact of such a change. The impact of changing the application owner's password should be fully evaluated before the application can be moved into production.
- The UTL_FILE_DIR directory. If you establish a UTL_FILE_DIR directory (as required for Log Miner, for example) then every user in the database can read from every file in the directory. Is that acceptable?
- RMAN/RMAN in the recovery catalog database. If you store your recovery catalog in a database instead of in the control files, then a user may be able to access your recovery catalog database. A user who can access your recovery catalog can edit or delete your recovery information. Choose an appropriately secure password for your RMAN account.

If you run batch programs from the command line, then other users on the system may be able to see your password via the ps -ef command (on UNIX). You can avoid that problem by using a 'here document', usually signaled via the use of <<. For example, the following shell script will not display the username and password via the ps command:

```
sqlplus <<EOF
system/mgr434543
@myscript
EOF
```

ATTACK 3: THANKS FOR TELLING ME WHERE TO LOOK

As of Net*8 V2, there is no need for any end user to need to know the name of any instance. You can use service names and aliases to mask the physical location and instance name for every database. The physical location of a database should be revealed solely on a need-to-know basis - and most people do not need to know.

If you use a tnsnames.ora file to manage access to database services, you should monitor its contents. When you distribute the tnsnames.ora file, you are allowing every user to read the file. The tnsnames.ora file may contain information about many databases, even though the users may only need to know about two or three databases. For example, your developers may only need access to services DEV1 and DEV2. Why distribute to them a file that also contains the database connection information for the PROD and HR services?

Maintaining multiple versions of your tnsnames.ora files may seem difficult at first. If your users are easily segregated by business process then you can divide the tnsnames.ora file into multiple files that will support each business division. Restrict the knowledge of the most sensitive databases to the group that requires it. If you avoid tnsnames.ora by using Oracle Names, then the SYSTEM account password is stored in the names.ora file. You must secure the names.ora file to prevent unauthorized access to this data.

ATTACK 4: THANKS FOR TELLING ME WHO HAS SYSDBA PRIVILEGE

Users with SYSOPER and SYSDBA privilege are listed in the V$PWFILE_USERS view. If you can gain access to the database, and cannot access the application owner accounts, you can query V$PWFILE_USERS to determine which users have the most privileges. If you use ORAPWD, you must secure your password file.

ATTACK 5: THANKS FOR LETTING ME IN

If you allow development and testing to be performed in your production database, you have several problems:

1. System performance of the production application will be impacted by the development and testing activity.
2. Change control is seldom well-defined; when is an application component in "development" or "test"? When is it moved to production?
3. Access is not well controlled. Developers should never be allowed full access to the production server unless the developers are also performing a well-defined change control and production process. If the developers have full access to the database and server, they have full access to your most sensitive data.

The last of these problems directly impacts your security plan. If development and testing occurs on a separate server, then the developers do not need accounts on the production server or in the production database. Your goal should be to remove all non-production users from the database and server.

ATTACK 6: THANKS FOR TURNING OFF THE RADAR

If you do not move the SYS.AUD$ table to a non-SYSTEM tablespace, then you will need to closely monitor the size of the audit log table. In many environments, auditing is turned off because of the space requirements of the AUD$ table. If the AUD$ table is left in the SYSTEM tablespace and you audit all login attempts, then a "ping flood" could disable your entire database by filling AUD$.

If you use Oracle's auditing features, you need to query the AUD$ information regularly. The data in AUD$ reflects events that have already happened. In order manage your security effectively, you need to have as small a gap as possible between the time the event occurs and the time you react to it. Select the events that you care the most about (such as unsuccessful accesses to the most sensitive tables) and report their audit results frequently.

From a database administration perspective, you can watch for potential suspicious activity by creating tables with misleading names. For example, you can create tables that are not part of the application, but which have names such as SALARY_HIST or LEDGER_CHECKS. If anyone accesses the dummy tables, their access should cause the writing of an audit record. Document and follow the procedures for reporting and reacting to security violations detected by your auditing procedures.

ATTACK 7: THANKS FOR THE PUBLIC DATABASE LINK

A database link supports access into another database. Access via database links should be tightly controlled. Within the database that owns the data, you should create an account that is used only by the database link. You should grant that account as few privileges as possible. You may need to create special views that are accessed only by that account.

For example, you may have a table named EMPLOYEE in your database. Another database, from the Dallas office, needs to access the EMPLOYEE table. Within your database, create a view on the EMPLOYEE table:

```
create user Dallas_Link identified by dfadsf8865
default tablespace users temporary tablespace temp;

grant create session to Dallas_Link;
```

From the AppOwner account:

```
create view Dallas_Employee as
select * from EMPLOYEE
where Location = 'DALLAS'
with check option;

grant select on Dallas_Employee to Dallas_Link;
```

Note that you can use the WITH READ ONLY clause of the CREATE VIEW command to further restrict users' access to the underlying tables.

From Dallas_Link account:

```
create synonym EMPLOYEE for Appowner.Dallas_Employee;
```

When someone uses the link, the user will enter the database as the Dallas_Link user, and access the only object accessible - the Dallas_Employee view. Unless you can control your data this tightly, you should not allow public database links to your production databases. Instead, you should use private database links into specific accounts so you can identify who accessed your database.

Consider the alternative: in the database that owns the EMPLOYEE table, you could create an account called Select_Link that has full SELECT privilege on the EMPLOYEE table. From a separate database, you could create a public database link that accesses the Select_Link account. Who is using that link? If you can access the Select_Link account, you can have access to the EMPLOYEE table with little auditing possible.

You may wish to use the PRODUCT_USER_PROFILE tables to further limit the ability of the Dallas_Link user to perform any actions within your database. For example, you can prevent that user from issuing any connect, host, or insert commands from within SQL*Plus.

Database link management is made more difficult by passwords that are subject to frequent changes. Every time you change the password of the Dallas_Link account, you will need to drop and re-create the database link. You will need to factor these changes into your administration procedures for the database.

If you use MTS, then you may choose to use shared database links to pool the connection resources required by database links. If you use a shared database link, you must provide an authentication account, as shown in the following example:

```
create shared database link HQ_LINK_SHARED
connect to current_user
authenticated by DUMMY
```

```
identified by DUMMYSPASS346314
using 'hq';
```

When you use the shared link, the authentication information in the link is used to verify your connection. You should not use an application or user account for the shared link authentication. Instead, create a separate account (called DUMMY in this example). Give that account CREATE SESSION privilege only. Do not use that account for any other purpose, and do not grant it any unnecessary privileges. Make that account's password difficult, since no one will ever type it in.

ATTACK 8: THANKS FOR NOT CHANGING YOUR PASSWORDS

As of Oracle8, you can force passwords to expire and you can prevent the reuse of old passwords. Be careful implementing this feature for the default profile, since you will impact every account that uses the default profile (including SYS, SYSTEM, and the application owner accounts). You will also need to provide users with a simple way to change their passwords.

The following script, which creates the CHANGE_MY_PASSWORD, provides a way for users to change their passwords via procedural calls. The only parameter for the procedure is the new password for the user. You can execute this stored procedure from within your application the same way you execute other stored procedures for your application. Thus, your users do not have to know the syntax for the alter user or password commands; they only need to provide the new password. For example, you may create a front-end form for a client-server application to prompt the user for a new password and then automatically execute the CHANGE_MY_PASSWORD procedure.

The CHANGE_MY_PASSWORD procedure executes the alter user command, which is a DDL command. Note that the alter user command does not perform the same password validation procedures as the password command. In order to execute a DDL command via PL/SQL, you need to use dynamic SQL.

The CHANGE_MY_PASSWORD procedure will only change the password for the current user; the User pseudocolumn provides the username value to the procedure. The procedure must be created under an account that has the ALTER ANY USER system privilege.

```
create or replace procedure CHANGE_MY_PASSWORD(NewPass IN VARCHAR2) AS
    Cursor_Name INTEGER;
    String VARCHAR2(100);
    cursor USERNAME_CURSOR is
      select User from DUAL;
    UserNm VARCHAR2(32);
BEGIN
   open USERNAME_CURSOR;
   fetch USERNAME_CURSOR into UserNm;
   close USERNAME_CURSOR;
   String:= 'alter user '||UserNm||' identified by '||NewPass ;
   Cursor_Name := DBMS_SQL.OPEN_CURSOR;
   DBMS_SQL.PARSE(Cursor_Name, String, dbms_sql.Native);
   DBMS_SQL.CLOSE_CURSOR(Cursor_Name);
END;
/
```

When you execute the CHANGE_MY_PASSWORD procedure, you pass the new password value as shown in the following example execution.

```
execute CHANGE_MY_PASSWORD('myn3wp');
```

If your password passes the password management criteria defined by the profile established for your account, ORACLE will respond to that command with the following message:

```
PL/SQL procedure successfully completed.
```

Note that if you use the password history features, Oracle will store the old encrypted passwords in the USER_HISTORY$ table in the SYSTEM tablespace. If you set a high value for the reuse time, then you may need to allocate additional space to the SYSTEM tablespace to support user password changes.

THWARTING ATTACKS

As noted in the preceding sections, each attack can be prevented or detected. As your database grows in size, you need to maintain the same level of attention to security. As your data is increasingly published to a wider audience, you need to make sure you apply the same level of "need to know" security criteria to your users. You can simplify the security management for a growing database by establishing the following:

1. Isolation of the production environment. Do not use the server or database for anything other than the production database.
2. Removal of unneeded accounts.
3. Frequent changes of passwords for sensitive accounts such as application object owners.
4. Well-defined, restricted database links.
5. Auditing of selected events, with frequent reporting of the accesses you care most about.
6. Restricted distribution of information about the servers, databases, and object names used.
7. Elimination of operating-system level security holes such as the SUID bit issues.

There are additional security options available, such as client-to-server encryption of passwords and the Oracle Security Server. Without having the preceding steps in place, there is little benefit to be gained by implementing additional complex security tools. If you've already given users a copy of the data, or removed all the access controls to privileged accounts, placing additional security controls on the network won't hinder the unauthorized access to the data.

With these basic steps in place, along with well-defined change management and production control procedures, you can provide a reasonable level of security as your database grows and your user base expands. You may need to develop utilities to support the specific needs of your users (such as customized tnsnames.ora files) in order to fully implement these solutions.