CRC Press
Taylor & Francis Group
AN AUERBACH BOOK

HOWTO
Secure and Audit Oracle 10g and 11g

Ron Ben Natan

Foreword by Pete Finnigan

Chapter 6

Authentication

Authentication comes from the Greek word *αυθεντικός*, which means real or genuine and "from the author" (authentes). Authentication is the process in which Oracle attempts to identify your identity when you logon to the database and decides whether to allow it. Authentication is the first step in any access to the database—it is the front gate to your castle. If you have no way to identify a party making the connection or there is a way to attack the authentication scheme and fake an identity, then nothing else matters.

In this chapter you'll review the various methods Oracle employs to authenticate users. Starting with the basic/standard authentication method using usernames and passwords, through password stores and password files, and all the way to advanced authentication options, you'll learn that you have many options for authenticating your users. You'll learn about methods in which the authenticating entity is the database as well as schemes in which the database delegates authentication to an external server. All of these options allow you to choose the most appropriate authentication scheme for your database environments or for different users.

6.1 HOWTO Understand and Use O3/O5 LOGON and OS Authentication

There are two main forms of basic authentication in Oracle—one in which authentication is done by the database and one in which Oracle trusts the operating system (OS) to perform the authentication of a user. In Chapter 4 you saw the basic working of the O3LOGON authentication scheme; instead of repeating it here you'll see what the equivalent method for Oracle 11g looks like. The authentication scheme in Oracle 11g is called O5LOGON.

O5LOGON

The changes in O5LOGON include the use of secure hash algorithm-1 (SHA-1), the use of salt and a longer session key. O5LOGON looks like this (these are segments from the network packets sent between the client and the server in a O5LOGON authentication process):

1. Logon to the database using one of the service names defined within your tnsnames.ora:

```
[oracle11@o11g ~]$ sqlplus scott/tiger@o11g
SQL*Plus: Release 11.1.0.6.0 - Production on Fri Nov 16 16:25:55 2007
Copyright (c) 1982, 2007, Oracle. All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Oracle Label Security, OLAP, Data Mining,
Oracle Database Vault, and Real Application Testing options
SQL>
```

2. The client communicates with the server and sends it the username (scott in this case):

```
0x0000:  4500  0128  95cf  4000  4006  6684  c0a8  de15  E..(..@.@.f.....
0x0010:  c0a8  de15  802a  05f1  8082  8a04  806c  8dad  .....*.........l.....
0x0020:  8018  0100  3e97  0000  0101  080a  003d  1399  ..... > ..... =.....
0x0030:  003d  1394  00f4  0000  0600  0000  0000  0376  .= .....v
0x0040:  02fe  ffff  ff05  0000  0001  0000  00fe  ffff  ..........
0x0050:  ff05  0000  00fe  ffff  fffe  ffff  ff05  7363  ..........sc
0x0060:  6f74  740d  0000  000d  4155  5448  5f54  4552  ott.....AUTH_TER
0x0070:  4d49  4e41  4c05  0000  0005  7074  732f  3100  MINAL.....pts/1.
0x0080:  0000  000f  0000  000f  4155  5448  5f50  524f  ........AUTH_PRO
0x0090:  4752  414d  5f4e  4d25  0000  0025  7371  6c70  GRAM_NM%...%sqlp
0x00a0:  6c75  7340  6f31  3167  2e67  7561  7264  6975  lus@o11g.dbasecu
0x00b0:  6d2e  636f  6d20  2854  4e53  2056  312d  5633  r.com.(TNS.V1-V3
0x00c0:  2900  0000  000c  0000  000c  4155  5448  5f4d  )........AUTH_M
0x00d0:  4143  4849  4e45  1100  0000  116f  3131  672e  ACHINE.....o11g.
0x00e0:  6775  6172  6469  756d  2e63  6f6d  0000  0000  dbasecur.com.....
0x00f0:  0800  0000  0841  5554  485f  5049  4404  0000  .....AUTH_PID...
0x0100:  0004  3430  3432  0000  0000  0800  0000  0841  ..4042.........A
0x0110:  5554  485f  5349  4408  0000  0008  6f72  6163  UTH_SID.....orac
0x0120:  6c65  3131  0000  0000                          le11.....
```

3. The server responds with the AUTH_SESSKEY and the AUTH_VFR_DATA. The AUTH_SESSKEY has been extended to 48 bytes in O5LOGON. The AUTH_VFR_DATA is the password hash salt value of length 10 bytes:

```
0x0000:  4500  0192  9ea7  4000  4006  5d42  c0a8  de15  E.....@.@.]B.....
0x0010:  c0a8  de15  05f1  802a  806c  8dad  8082  8af8  .....*.l.....
0x0020:  8018  0100  3f01  0000  0101  080a  003d  13a6  ....?.....=.....
0x0030:  003d  1399  015e  0000  0600  0000  0000  0803  .= .....^.....
0x0040:  000c  0000  000c  4155  5448  5f53  4553  534b  ......AUTH_SESSK
0x0050:  4559  6000  0000  6039  3443  3439  3735  3938  EY`.....'94C497598
0x0060:  3445  4537  3441  3345  3739  4339  4538  3546  ...4EE74A3E79C9E85F
0x0070:  4446  3038  3542  3430  4236  3444  3234  4333  ...DF085B40B64D24C3
0x0080:  4138  4234  4239  4533  3741  4138  4546  3434  ...A8B4B9E37AA8EF44
0x0090:  3832  4544  4139  3636  4234  3446  3136  3843  ...82EDA966B44F168C
0x00a0:  4338  4534  4534  3133  4242  4138  4337  3438  ...C8E4E413BBA8C748
0x00b0:  3932  3130  3233  3600  0000  000d  0000  000d  9210236.........
0x00c0:  4155  5448  5f56  4652  5f44  4154  4114  0000  AUTH_VFR_DATA.....
0x00d0:  0014  3242  3233  3445  3434  3437  3832  3541  .....2B234E4447825A
0x00e0:  3537  3046  3144  251b  0000  1a00  0000  1a41  570F1D%.........A
0x00f0:  5554  485f  474c  4f42  414c  4c59  5f55  4e49  UTH_GLOBALLY_UNI
0x0100:  5155  455f  4442  4944  0020  0000  0020  3434  QUE_DBID..... 44
0x0110:  3336  3339  3935  3432  4142  3630  3539  4645  ...36399542AB6059FE
0x0120:  3134  3432  3832  3743  3834  3144  4433  0000  1442827C841DD3.....
```

```
0x0130:  0000  0401  0000  0002  0001  0000  0000  0000  ................................
0x0140:  0000  0000  0000  0000  0000  0000  0000  0000  ................................
0x0150:  0000  0000  0000  0000  0000  0000  0000  0000  ................................
0x0160:  0000  0002  0000  0000  0000  3601  0000  0000  ..........................6.....
0x0170:  0000  0055  550f  0000  0000  0000  0000  0000  ...............UU.............
0x0180:  0000  0000  0000  0000  0000  0000  0000  0000  ................................
0x0190:  0000                                             ..
```

The 10 byte salt value is sent to the client because in Oracle 11g the password hashing scheme is based on SHA-1 with a salt component added (some random bits used to strengthen the password in the face of brute-force attack). The client will need to use the same salt value that the server uses in order for the match to occur. In Oracle 11g the password hash is created by concatenating the salt component to the password. SHA-1 is then applied to the salted password. The first 20 bytes are converted to printable hex—and this is the password hash that is stored in SPARE4 of SYS.USER$. The salt is also stored in SPARE4:

```
SQL> select name,spare4 from user$ where name='SCOTT';
NAME
------------------------------
SPARE4
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
------------------------------
SCOTT
S:D520818940AE5CA8F426B1C3EC408AB446C203AA2B234E4447825A570F1D
```

As you can see, there are 30 bytes stored in SPARE4 just after "S:". They are stored as printable hex so there are 60 hex characters. The first 40 characters are the 20 bytes from the SHA-1 result and the last 20 characters come from the ten salt bytes. The salt (`2B234E4447825A570F1D`) is sent in the packet shown above as the AUTH_VFR_DATA.

4. The client takes the salt that was sent by the server and computes the hash for the salted password using SHA-1. It then uses this value to decrypt the session key. The session key was encrypted by the server using AES192 using the hash—so the client performs the inverse operation. The decryption key is derived by taking the first 20 bytes of the SHA-1 (i.e., 160 bits) and adding four zeros (to make 192 bits). The client then creates its own session key. The two AUTH_SESSKEY values are combined by the client to form the key that is then used for password encryption. Then, the client sends both its encrypted session key (AUTH_SESSKEY below) and the encrypted password (AUTH_PASSWORD) to the server:

```
0x0000:  4500  03bf  95d1  4000  4006  63eb  c0a8  de15  E.....@.@.c.....
0x0010:  c0a8  de15  802a  05f1  8082  8af8  806c  8f0b  .....*.....l.....
0x0020:  8018  0100  412e  0000  0101  080a  003d  13ad  ....A.....  =.....
0x0030:  003d  13a6  038b  0000  0600  0000  0000  0373  .=  .....s
0x0040:  03fe  ffff  ff05  0000  0001  0100  00fe  ffff  ................................
0x0050:  ff12  0000  00fe  ffff  fffe  ffff  ff05  7363  ..............................sc
0x0060:  6f74  740c  0000  000c  4155  5448  5f53  4553  ott.....AUTH_SES
0x0070:  534b  4559  6000  0000  fe40  4231  3338  4639  SKEY`.....@B138F9
0x0080:  4246  4635  4632  3333  4437  3534  4632  3130  BFF5F233D754F210
0x0090:  3230  3143  4631  3637  3645  3433  4238  3435  201CF1676E43B845
0x00a0:  4139  4146  4242  4445  3646  3845  4245  4545  A9AFBBDE6F8EBEEE
0x00b0:  4643  3645  3842  4544  4443  2041  3841  4234  FC6E8BEDDC.A8AB4
0x00c0:  3746  3145  3132  3230  4141  3333  3335  3946  7F1E1220AA33359F
0x00d0:  3032  3844  4645  3633  3345  3400  0100  0000  028DFE633E4.....
0x00e0:  0d00  0000  0d41  5554  485f  5041  5353  574f  .....AUTH_PASSWO
```

```
0x00f0:   5244   4000   0000   4046   3043   3845   4543   3039   RD@.....@F0C8EEC09
0x0100:   3843   3644   3335   3836   3946   3830   4346   3738   8C6D35869F80CF78
0x0110:   4432   3144   3246   3736   3146   3244   3446   3743   D21D2F761F2D4F7C
0x0120:   4543   3838   3144   4632   4446   4131   3033   4544   EC881DF2DFA103ED
0x0130:   3430   3534   4536   4300   0000   0008   0000   0008   4054E6C.........
```

5. The server receives both values. It decrypts the session key sent from the client using AES192 and the password hash in the same way that the client decrypted the server's session key. Now the server has both the server session key and the client session key so it can go ahead and decrypt the password and check whether or not to authenticate the user.

O5LOGON improves on O3LOGON but it also uses a username and a password, password hashes, etc. In all these authentication methods, the usernames and passwords are kept within the data dictionary and it is the Oracle server that does the authentication. Although these are by far the most common ways that you might be authenticated to an Oracle server, you can also configure Oracle to trust the OS to authenticate you on its behalf.

Operating System Authentication

In OS authentication, the user authenticates at the OS level and then connects to the database. When connecting to the database the user does not have to supply a password and Oracle maps the database user to the OS user based on its name. Privilege management is still done by the database—only the authentication is delegated to the OS.

Oracle distinguishes between two cases—when the user is logged onto the local OS and when the user is connecting over Oracle*Net. When a user is already logged onto the local OS, Oracle only needs to know how to map the OS credentials to its credentials and it will trust the OS. Trusting a remote OS is one of the worst things you can do in terms of security and in fact, this has been deprecated in Oracle 11g. The risk is that someone can just put up a node on the network, configure a user that maps to a database administrator (DBA), and gain entry with no authentication being done on the server.

To enable remote OS authentication on versions prior to 11g you need to set the initialization parameter REMOTE_OS_AUTHENT:

```
SQL> show parameter remote_os_authent;

NAME                          TYPE                  VALUE
----------------------------  --------------------  -------------------

remote_os_authent             boolean               FALSE
SQL> alter system set remote_os_authent=true scope=spfile;
System altered.
SQL> shutdown
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area      418484224 bytes
Fixed Size                    1300324 bytes
Variable Size                 268437660 bytes
Database Buffers              142606336 bytes
Redo Buffers                  6139904 bytes
Database mounted.
Database opened.
```

```
SQL>show parameter remote_os_authent;

NAME                          TYPE                  VALUE
----------------------        ----------           ----------
remote_os_authent             boolean               TRUE
```

At this point, Oracle is able to accept also remote users authenticated by the client OS. You do not need to set that to have users logged onto the local OS to gain access—for that you just need to define a user as IDENTIFIED EXTERNALLY.

This does not mean you can't authenticate to Oracle using a username and a password. For users not defined as IDENTIFIED EXTERNALLY, Oracle still performs the authentication:

```
SQL> connect scott
Enter password:
Connected.
```

When using OS authentication you need to decide how to map OS usernames to Oracle usernames. The initialization parameter OS_AUTHENT_PREFIX controls that mapping. The database username that will be used within the database is the concatenation of this value along with the OS username. Before Oracle 11g the default for this parameter was ops$. In 11g it is set to be null and this is a more secure setting which you should adopt for pre-11g servers. It is also the value that is set once you install Oracle Database Vault as part of general hardening that Database Vault performs. If this value is set to ops$ and you are logged onto the OS as jane then your logon to the database will be as user ops$jane. If this value is set to null then you will log onto the database as user jane.

If you enable remote OS authentication and have (as an example) the prefix set to ops$, you can allow access from OS users by creating a user IDENTIFIED EXTERNALLY:

```
SQL> create user ops$jane identified externally;
User created.
SQL> grant create session to ops$jane;
Grant succeeded.
```

Now logon to the OS as jane:

```
% su - jane
```

You don't need to specify a username or a password to logon to Oracle:

```
saturn:oracle10% sqlplus /
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jan 25 11:09:05 2008
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, and Data Mining options
SQL> show user
USER is "OPS$JANE"
```

Why did the default for this parameter change from ops$ to a null value? The reason has to do with confusion that can happen when you have a nontrivial mapping. The ops$ default was originally created so it would be clear which users were authenticated by Oracle and which users were authenticated by

the OS. However, it allowed a situation in which you have both a user called OPS$JANE within the database and a user called JANE within the database. Both are valid and different—but you can see that it can be very confusing. Once you change the parameter value to none, you avoid this confusion—but also lose the indication within the username of the authentication. If you are going to use ops$ (or any non-null mapping value) you should adopt a practice by which you create a user with an impossible password (see Section 4.7) for each user with the ops$ prefix—e.g.,

```
SQL> create user jane identified by impossible;
User created.
```

Administrator Authentication

One question that you may be asking is whether you already use OS authentication when you connect "/ as sysdba"? The answer is that this is indeed a form of Oracle trusting a user that has been authenticated with the OS but this mode is called administrator authentication and is quite different. You have special handling for administrator authentication to solve the chicken-and-egg problem of starting a database. If the normal authentication method is O3/O5 LOGON and the database does the authentication using usernames and passwords stored in the database, how do you authenticate the person who needs to start the database? If the database is down, how is it going to authenticate this administrator and check their password? To solve this, Oracle authenticates administrators (users who have the sysdba or sysoper system privileges) using the OS separately from IDENTIFIED EXTERNALLY.

In almost all installations of Oracle, if you are logged onto the OS as the Oracle instance account you can connect as the SYS user using sysdba or sysoper privileges without supplying a password:

```
oracle10% sqlplus "/ as sysdba"
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jan 25 11:59:41 2008
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, and Data Mining options
SQL> show user
USER is "SYS"
```

To understand how this works, let's look at Oracle on Unix. When you install Oracle on Unix there are two OS groups, OSDBA and OSOPER, that map into two Unix user groups—dba and oper. When an OS user tries to log onto Oracle as /, Oracle checks the Unix user group for which the user belongs to. If the user belongs to the Unix user group dba and the connection is being done as sysdba then Oracle will allow the connection and log the user on as SYS with the sysdba privileges. If the user belongs to the Unix user group oper and the connection is being done as sysoper, then Oracle will allow the connection and log the user on as PUBLIC with the sysoper privileges:

```
oracle10% sqlplus "/ as sysoper"

SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jan 25 12:04:39 2008
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, and Data Mining options
SQL> show user
USER is "PUBLIC"
```

In both these cases there is no explicit username or password. This is a form of OS authentication—but it requires you to be logged in locally to the host where the database is installed and it relies on Unix user groups. It is not a general-purpose authentication scheme that you can use for any user. Also, it cannot be used for connecting as sysdba or sysoper over the network—for that you need to set up password files.

Two Things to Remember about OS Authentication

1. Prefer to use a null REMOTE_AUTHENT_PREFIX.
2. If you are using OPS$ (due to historical reasons) make sure that for every externally identified account by the name OPS$ < USER > you create a regular account < USER > with an impossible password.

6.2 HOWTO Use Password Files

As you saw in the previous HOWTO, one way to logon using sysdba and sysoper privileges is by signing onto the OS as the instance owner in which case you can logon with no password. But what happens if you want to sign on with sysdba or sysoper privileges over the network or from another account that perhaps is not the one that belongs to the special Unix user groups? Then—you can use password files.

Password files allow you to set passwords that are stored outside the database and that are used for authenticating administrators. These passwords are stored in an external file that is encrypted by Oracle. They can be used even if the database is down—so even if you logon and need to start the database you can do so and have Oracle authenticate you with a password.

To create a password file use the ORAPWD utility:

```
$ orapwd file=./11g_pwd entries=100 ignorecase=n
Enter password for SYS:
```

When you run orapwd Oracle asks you for the SYS password. It creates a file based on the file name you provide for the parameter. Provide a full path name or a relative path to your current working directory. You can define the maximal number of entries that the file may contain. You can also specify whether you want passwords in the password file to be case sensitive or case insensitive (this feature is new in 11g).

Creating the password file is not enough. You also have to set the REMOTE_LOGIN_PASSWORDFILE initialization parameter to an appropriate value (by default, the value allows you to use password files). Available values for this parameter are

- NONE—causing Oracle to behave as though a password file does not exist.
- EXCLUSIVE—which is the default, causes Oracle to use the password files if connections with sysdba or sysoper are attempted other than from the local server. EXCLUSIVE means that the password file is being used only by your database and that you can modify it from within the database (whenever, e.g., you grant such privileges to an administrator).
- SHARED—allows you to use a single password file for multiple databases, but none of them can update the password file. If you need to update the password file then you need to switch this parameter to EXCLUSIVE in one of the databases, change the password file and then change it back to be used as SHARED.

Now you can connect with sysdba or sysoper privileges even beyond the security of the instance account. For example, you can be logged onto another machine and connect as sysdba:

```
$ sqlplus sys@o11g as sysdba
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Nov 16 02:52:10 2007
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Enter password:
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, Oracle Label Security, OLAP, Data Mining,
Oracle Database Vault, and Real Application Testing options
SQL>
```

Passwords are added as needed to the password file. A password needs to be added whenever you grant such a privilege to a user. For example, if you GRANT SYDBA to a user, Oracle must write the new password to the file so that this user can logon using the password file. As an example, the password file will be updated when you perform the following grant:

```
SQL> grant sysdba to ronb;
Grant succeeded.
```

If you have to re-create your password file you have to write all passwords for users with these privileges out to the password file. For each user it is enough to regrant the privilege. To know to which users you need to regrant the privilege, use the V$PWFILE_USERS view. As an example, if you see the following in the view:

```
SQL> select * from v$pwfile_users;

USERNAME            SYSDB SYSOP SYSAS
---------           ------ ------ ------

SYS                 TRUE  TRUE  FALSE
JANE                TRUE  FALSE FALSE
RONB                TRUE  FALSE FALSE
```

then you can re-create the passwords in the password file by performing

```
SQL> grant sysdba to sys;
Grant succeeded.
SQL> grant sysdba to jane;
Grant succeeded.
SQL> grant sysdba to ronb;
Grant succeeded.
```

Two Things to Remember about Password Files

1. Prefer password files for sysdba and sysoper connections.
2. Use shared password files only after you review the process you will need to perform to change a password and ensuring that you can perform this without bringing any of your production databases down.

6.3 HOWTO Configure Clients to Use External Password Stores

In Chapter 5 you learned about the Oracle wallet. In addition to storing keys, certificates, and certificate requests, the wallet can be used as an external password store for allowing you to authenticate with an Oracle server without supplying a password at logon time. The wallet serves as a "central container of secrets" for Oracle environments and as such it is a natural place to put passwords. When you put a password in a wallet you can be authenticated to an Oracle database without manually supplying a password—the password is taken from the wallet where it is stored. This is a very convenient and secure option that allows you, among other things, to have batches and scripts connect to the database without having to embed passwords inside these batches, a dangerous security hole.

To enable connections through an external password store (in a wallet) follow these steps:

Step 1: Create an auto-login wallet:

```
$ mkstore -wrl./oracle/product/10.2.0/db_1/wallets/ -create
Enter password:
Enter password again:
```

You need to enter a password that protects the wallet but the wallet you create is an auto-login wallet so that you can use it without embedding a password in the script. The password is needed to create, change, or delete credentials but is not needed to make the connection. The wallet used is the .sso wallet (an auto-login wallet):

```
$ ls -l./oracle/product/10.2.0/db_1/wallets/
total 24
-rw------- 1 oracle10 dba 7940 Jan 25 03:52 cwallet.sso
-rw------- 1 oracle10 dba 7912 Jan 25 03:52 ewallet.p12
```

Step 2: Once you have the sso wallet in place, you can create a secure credential within the wallet. You need to give mkstore the username you will be using to log on and the service name that you use. For example, suppose you want to connect to the database on0rh4 that appears in your tnsnames.ora as follows:

```
on0rh4 =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.222.128)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = on0r4231.guardium.com)
  )
)
```

If you want to be able to connect as the user system in this database without supplying a password, insert the password into the wallet by running:

```
$ mkstore -wrl./oracle/product/10.2.0/db_1/wallets -createCredential '
on0rh4' 'system' <your password here>
Enter password:
Create credential oracle.security.client.connect_string1
```

You must see the message `Create credential oracle.security.client.connect_string1` in the output—that means that your credentials were created. The password that you enter is the

wallet password—not the account password to the database you want to connect to. The account password is passed on the command line (which is a bit of a problem from a security perspective). If you don't supply the password on the command line the credentials will not be created and you will get an error message:

```
$ mkstore -wrl ./oracle/product/10.2.0/db_1/wallets/ -createCredential
on0rh4 system
Enter password:
Argument needed for command: -createCredential
```

You can check the size of your files to make sure new credentials were created:

```
$ ls -l ./oracle/product/10.2.0/db_1/wallets/
total 32
-rw------- 1 oracle10 dba 8308 Jan 25 19:46 cwallet.sso
-rw------- 1 oracle10 dba 8280 Jan 25 19:46 ewallet.p12
```

You can also list the credentials to make sure they are within the wallet:

```
$ mkst\ore -wrl ./oracle/product/10.2.0/db_1/wallets -
listCredential
Enter password:

List credential (index: connect_string username)
1: o11g system
2: on0rh4 system
```

Step 3: Edit sqlnet.ora to tell SQL*NET that it should use the wallet as an external password store, and that it should not be using Secure Sockets Layer (SSL) authentication (which we'll cover later). You need to tell Oracle where to look for the wallet from which credentials should be taken:

```
SQLNET.WALLET_OVERRIDE = TRUE
SSL_CLIENT_AUTHENTICATION = FALSE

WALLET_LOCATION =
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
  (DIRECTORY = /home/oracle10/oracle/product/10.2.0/db_1/wallets/)
  )
)
```

The first line tells SQL*NET that any connection of the form /@connection_string should cause a lookup for the credentials in the wallet.

Step 4: Bounce the listener:

```
$ lsnrctl stop
LSNRCTL for Linux: Version 10.2.0.1.0 - Production on 25-JAN-2008 20:13:49
Copyright (c) 1991, 2005, Oracle. All rights reserved.
Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=ORCL)))
```

```
The command completed successfully
$ lsnrctl start
LSNRCTL for Linux: Version 10.2.0.1.0 - Production on 25-JAN-2008 20:13:59
Copyright (c) 1991, 2005, Oracle. All rights reserved.
Starting /home/oracle10/oracle/product/10.2.0/db_1/bin/tnslsnr: please wait…
…
The command completed successfully
```

At this point you can connect to the remote database without supplying a password:

```
$ sqlplus /@on0rh4
SQL*Plus: Release 10.2.0.1.0 - Production on Fri Jan 25 20:20:31 2008
Copyright (c) 1982, 2005, Oracle. All rights reserved.
Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
With the Partitioning, OLAP, and Data Mining options
SQL>
```

If you need to change the password stored within the wallet:

```
$ mkstore -wrl ./oracle/product/10.2.0/db_1/wallets -
modifyCredential o11g system <your password here>
Enter password:
Modify credential
Modify 1
```

If you need to delete a credential from the wallet:

```
$ mkstore -wrl ./oracle/product/10.2.0/db_1/wallets -deleteCredential o11g
Enter password:
Delete credential
Delete 1
$ mkstore -wrl ./oracle/product/10.2.0/db_1/wallets -listCredential
Enter password:
List credential (index: connect_string username)
2: on0rh4 system
```

As Listing 6.1 shows, using an external store only affects the fact that you do not supply a password on the connection—the authentication scheme is unchanged (in this case still using O3LOGON) and the logon is not encrypted. In the next HOWTO, you'll see another scheme that makes a more fundamental change.

Two Things to Remember about External Password Stores

1. Wallets are used to store all secrets—not only certificates. An example is the use of a wallet to store a password.
2. Use an auto-login wallet to store passwords so that your scripts don't have to include database passwords in cleartext. Embedding passwords is a serious risk because attackers will often search through scripts.

```
0x0000:  4500 013d 2eab 4000 4006 ccbd c0a8 de80   E..=..@.@.......
0x0010:  c0a8 de80 8de5 05f1 4937 ee2e 49a2 176d   ........I7..I..m
0x0020:  8018 1000 3f82 0000 0101 080a 01b9 c08d   ....?...........
0x0030:  01b9 c08c 0109 0000 0100 0000 0139 012c   .............9.,
0x0040:  0c01 0800 7fff 7f08 0000 0100 00cf 003a   ...............:
0x0050:  0000 0200 4141 0000 0000 0000 0000 0000   ....AA..........
0x0060:  0000 0000 0000 0000 0000 0000 0000 2844   ..............(D
0x0070:  4553 4352 4950 5449 4f4e 3d28 4144 4452   ESCRIPTION=(ADDR
0x0080:  4553 533d 2850 524f 544f 434f 4c3d 5443   ESS=(PROTOCOL=TC
0x0090:  5029 2848 4f53 543d 3139 322e 3136 382e   P)(HOST=192.168.
0x00a0:  3232 322e 3132 3829 2850 4f52 543d 3135   222.128)(PORT=15
0x00b0:  3231 2929 2843 4f4e 4e45 4354 5f44 4154   21))(CONNECT_DAT
0x00c0:  413d 2853 4552 5645 523d 4445 4449 4341   A=(SERVER=DEDICA
0x00d0:  5445 4429 2853 4552 5649 4345 5f4e 414d   TED)(SERVICE_NAM
0x00e0:  453d 6f6e 3072 3432 3331 2e67 7561 7264   E=on0r4231.dbase
0x00f0:  6975 6d2e 636f 6d29 2843 4944 3d28 5052   cur.com)(CID=(PR
0x0100:  4f47 5241 4d3d 7371 6c70 6c75 7329 2848   OGRAM=sqlplus)(H
0x0110:  4f53 543d 7268 3475 3278 3332 702e 6775   OST=rh4u2x32p.db
0x0120:  6172 6469 756d 2e63 6f6d 2928 5553 4552   asecur.com)(USER
0x0130:  3d6f 7261 636c 6531 3029 2929 29          =oracle10)))))
```

```
0x0000:  4500 0134 2eb7 4000 4006 ccba c0a8 de80   E..4..@.@.......
0x0010:  c0a8 de80 8de5 05f1 4937 f150 49a2 18e7   ........I7.PI...
0x0020:  8018 1000 3f79 0000 0101 080a 01b9 c117   ....?y..........
0x0030:  01b9 c115 0100 0000 0600 0000 0000 0376   ...............v
0x0040:  02d0 a108 0806 0000 0001 0000 00d8 bcff   ................
0x0050:  bf05 0000 0080 b9ff bfa8 beff bf06 7379   ..............sy
0x0060:  7374 656d 0d00 0000 0d41 5554 485f 5445   stem.....AUTH_TE
0x0070:  524d 494e 414c 0500 0000 0570 7473 2f32   RMINAL.....pts/2
0x0080:  0000 0000 0f00 0000 0f41 5554 485f 5052   .........AUTH_PR
0x0090:  4f47 5241 4d5f 4e4d 2a00 0000 2a73 716c   OGRAM_NM*...*sql
0x00a0:  706c 7573 4072 6834 7532 7833 3270 2e67   plus@rh4u2x32p.d
0x00b0:  7561 7264 6975 6d2e 636f 6d20 2854 4e53   dbasecu.com.(TNS
0x00c0:  2056 312d 5633 2900 0000 000c 0000 000c   .V1-V3).........
0x00d0:  4155 5448 5f4d 4143 4849 4e45 1600 0000   AUTH_MACHINE....
0x00e0:  1672 6834 7532 7833 3270 2e67 7561 7264   .rh4u2x32p.dbase
0x00f0:  6975 6d2e 636f 6d00 0000 0008 0000 0008   cur.com.........
0x0100:  4155 5448 5f50 4944 0500 0000 0531 3735   AUTH_PID.....175
0x0110:  3233 0000 0000 0800 0000 0841 5554 485f   23.........AUTH_
0x0120:  5349 4408 0000 0008 6f72 6163 6c65 3130   SID.....oracle10
0x0130:  0000 0000                                 ....
```

```
0x0000:  4500 0115 d7ef 4000 4006 23a1 c0a8 de80   E.....@.@.#.....
0x0010:  c0a8 de80 05f1 8de5 49a2 18e7 4937 f250   ........I...I7.P
0x0020:  8018 1000 3f5a 0000 0101 080a 01b9 c120   ....?Z..........
0x0030:  01b9 c117 00e1 0000 0600 0000 0000 0802   ................
0x0040:  000c 0000 000c 4155 5448 5f53 4553 534b   ......AUTH_SESSK
0x0050:  4559 4000 0000 4030 3234 3245 3030 4431   EY@...@0242E00D1
0x0060:  4645 4432 4641 3332 3139 3039 3941 3938   FED2FA3219099A98
0x0070:  3443 3641 3842 4446 3233 4234 3142 3941   4C6A8BDF23B41B9A
0x0080:  3531 4130 3934 3433 4146 3646 3038 3442   51A09443AF6F084B
0x0090:  3233 3444 3043 4200 0000 000d 0000 000d   234D0CB.........
0x00a0:  4155 5448 5f56 4652 5f44 4154 4100 0000   AUTH_VFR_DATA...
0x00b0:  0039 0900 0004 0100 0000 0200 0100 0000   .9..............
0x00c0:  0000 0000 0000 0000 0000 0000 0000 0000   ................
0x00d0:  0000 0000 0000 0000 0000 0000 0000 0000   ................
```

Listing 6.1 Authentication sequence does not change when using an external password store.

```
0x00e0:  0000 0000 0000 0200 0000 0000 0036 0100    .............6..
0x00f0:  0000 0000 0020 13bd 0c00 0000 0000 0000    .................
0x0100:  0000 0000 0000 0000 0000 0000 0000 0000    .................
0x0110:  0000 0000 00                               .....
```

```
0x0000:  4500 02e4 2eb9 4000 4006 cb08 c0a8 de80    E.....@.@.......
0x0010:  c0a8 de80 8de5 05f1 4937 f250 49a2 19c8    ........I7.PI...
0x0020:  8018 1000 4129 0000 0101 080a 01b9 c123    ....A).........#
0x0030:  01b9 c120 02b0 0000 0600 0000 0000 0373    ...............s
0x0040:  03d0 a108 0806 0000 0001 0100 00fc e8ff    ................
0x0050:  bf0d 0000 00a4 e5ff bf90 f5ff bf06 7379    ..............sy
0x0060:  7374 656d 0c00 0000 0c41 5554 485f 5345    stem.....AUTH_SE
0x0070:  5353 4b45 5940 0000 0040 4642 3931 4334    SSKEY@...@FB91C4
0x0080:  3239 4346 3639 3742 3242 3739 4532 3142    29CF697B2B79E21B
0x0090:  3243 3841 3643 3733 3038 4543 4537 3442    2C8A6C7308ECE74B
0x00a0:  3146 4539 3546 4637 4438 4130 4545 3734    1FE95FF7D8A0EE74
0x00b0:  4236 3330 3436 3743 3043 0100 0000 0d00    B630467C0C......
0x00c0:  0000 0d41 5554 485f 5041 5353 574f 5244    ...AUTH_PASSWORD
0x00d0:  4000 0000 4039 4539 3345 4135 4430 4644    @...@9E93EA5D0FD
0x00e0:  4531 4435 4443 4637 4539 4137 4338 3938    E1D5DCF7E9A7C898
0x00f0:  4446 3933 3444 4130 4342 3137 3139 3035    DF934DA0CB171905
0x0100:  4338 3438 4642 4632 4446 3746 3941 3736    C848FBF2DF7F9A76
0x0110:  3744 3744 4600 0000 0008 0000 0008 4155    7D7DF.........AU
0x0120:  5448 5f52 5454 0500 0000 0531 3639 3630    TH_RTT.....16960
0x0130:  0000 0000 0d00 0000 0d41 5554 485f 434c    .........AUTH_CL
0x0140:  4e54 5f4d 454d 0400 0000 0434 3039 3600    NT_MEM.....4096.
0x0150:  0000 000d 0000 000d 4155 5448 5f54 4552    ........AUTH_TER
0x0160:  4d49 4e41 4c05 0000 0005 7074 732f 3200    MINAL.....pts/2.
0x0170:  0000 000f 0000 000f 4155 5448 5f50 524f    ........AUTH_PRO
0x0180:  4752 414d 5f4e 4d2a 0000 002a 7371 6c70    GRAM_NM*...*sqlp
0x0190:  6c75 7340 7268 3475 3278 3332 702e 6775    lus@rh4u2x32p.db
0x01a0:  6172 6469 756d 2e63 6f6d 2028 544e 5320    asecur.com.(TNS.
0x01b0:  5631 2d56 3329 0000 0000 0c00 0000 0c41    V1-V3).........A
0x01c0:  5554 485f 4d41 4348 494e 4516 0000 0016    UTH_MACHINE.....
0x01d0:  7268 3475 3278 3332 702e 6775 6172 6469    rh4u2x32p.dbasec
0x01e0:  756d 2e63 6f6d 0000 0000 0800 0000 0841    ur.com.........A
0x01f0:  5554 485f 5049 4405 0000 0005 3137 3532    UTH_PID.....1752
0x0200:  3300 0000 0008 0000 0008 4155 5448 5f53    3.........AUTH_S
0x0210:  4944 0800 0000 086f 7261 636c 6531 3000    ID.....oracle10.
0x0220:  0000 0008 0000 0008 4155 5448 5f41 434c    ........AUTH_ACL
0x0230:  0400 0000 0434 3430 3000 0000 0012 0000    .....4400.......
0x0240:  0012 4155 5448 5f41 4c54 4552 5f53 4553    ..AUTH_ALTER_SES
0x0250:  5349 4f4e 2500 0000 2541 4c54 4552 2053    SION%...%ALTER.S
0x0260:  4553 5349 4f4e 2053 4554 2054 494d 455f    ESSION.SET.TIME_
0x0270:  5a4f 4e45 3d27 2d30 353a 3030 2700 0100    ZONE='-05:00'...
0x0280:  0000 1700 0000 1741 5554 485f 4c4f 4749    .......AUTH_LOGI
0x0290:  4341 4c5f 5345 5353 494f 4e5f 4944 2000    CAL_SESSION_ID..
0x02a0:  0000 2034 3439 3545 3741 3835 3637 3744    ...4495E7A85677D
0x02b0:  3935 4445 3034 3041 3843 3038 3044 4534    95DE040A8C080DE4
0x02c0:  3437 3300 0000 0010 0000 0010 4155 5448    473.........AUTH
0x02d0:  5f46 4149 4c4f 5645 525f 4944 0000 0000    _FAILOVER_ID....
0x02e0:  0000 0000                                  ....
```

Listing 6.1 (continued)

6.4 HOWTO Configure SSL-Based Authentication Using ASO

Another authentication method supported by Oracle that uses the wallet is SSL authentication. This authentication method requires you to have Oracle Advanced Security Options (ASO) installed. This authentication method is more complex to set up than external password stores but is far more comprehensive and functional. SSL authentication makes use of certificates rather than passwords. The certificates are stored within the wallet and used when you make the connection. Like in the external password store HOWTO, this allows you to connect without supplying a password. However, unlike the previous HOWTO, SSL authentication buys you much more—authentication can happen from anywhere on the network, the communications is encrypted, and certificates are used for mutual authentication. This can come in handy when you want to ensure that only your application server makes a connection to the database—in which case you will place the certificate in the application server's wallet and thus guarantee that connections can only be made locally on the database server or from the application server.

Figure 6.1 shows what the handshake between the client and the server looks like when you use SSL authentication:

1. The client initiates an SSL connection using a TCPS entry in tnsnames. It does not need to logon with a password.
2. The client and the server negotiate a cipher suite, which they will use during the communications.
3. The server gets its certificate from the wallet and sends it to the client.
4. The client checks whether it can trust the server's certificate. It checks within the wallet whether this certificate is a trusted certificate or is signed by a trusted certificate authority (CA) (see Chapter 5).
5. If client authentication is required then the client gets its certificate from its wallet and sends it to the server.
6. The server goes through the same process to check whether it can trust the client's certificate.
7. The client and server use the public keys within the certificates to exchange random session keys that will be used as a symmetric key for network encryption.
8. At this point communication can begin—each side has authenticated and approved the other party and they have an encryption key and a cipher suite with which they can set up encrypted communication.

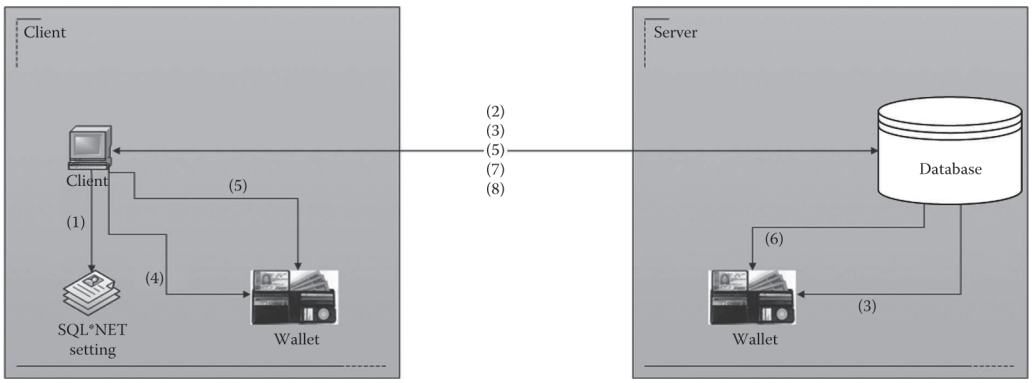

Figure 6.1 Initiating a connection using certificates for authentication.

Next, let's go through a complete setup process and see how to set up SSL authentication. This sequence includes the minimum setup requirements. In addition to this sequence you can control many things such as the available cipher suites, the prioritization of the cipher suites, etc. The basic steps are:

Step 1: First you need to create an auto-login wallet to store your certificates and in order for Oracle to be able to check whether certificates can be trusted. In Chapter 5 you learned how to use orapki to set up wallets and certificates; let's set things up using the Oracle Wallet Manager (OWM) this time. On the server, you already created an auto-login wallet in the previous HOWTO—so we'll use that wallet on the server. Start up the OWM:

```
$ owm
```

Click on Wallet → Open and browse your wallet location. Enter the password to the wallet to open it. At this point the wallet is empty—there are no certificates that you can use as shown in Figure 6.2. Verify that the wallet is indeed an auto-login wallet by clicking on the Wallet menu option—the second to last entry should have Auto Login checked. If it is not, check it.

Step 2: Edit sqlnet.ora and listener.ora on the server and add an entry to include the wallet location. The entry should look like the following and include a full path name:

```
WALLET_LOCATION =
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
  (DIRECTORY = /home/oracle10/oracle/product/10.2.0/db_1/wallets/)
  )
 )
```

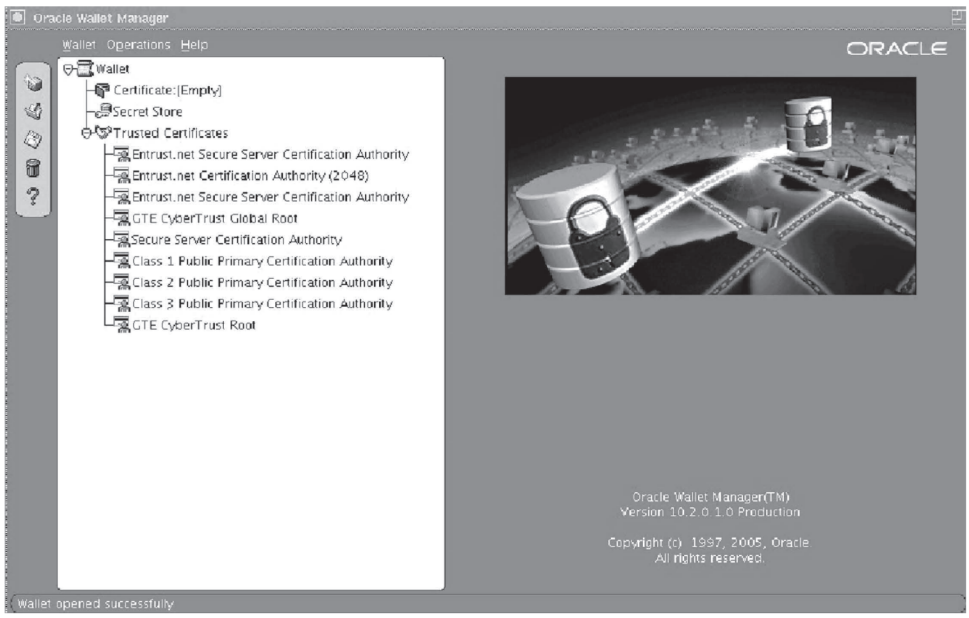

Figure 6.2 Viewing the wallet using OWM.

Step 3: Edit sqlnet.ora to define that the server should also authenticate the client (i.e., the default). You want the server to use the client certificate rather than require you to logon with a password:

```
SSL_CLIENT_AUTHENTICATION = TRUE
```

Step 4: Edit sqlnet.ora to add the SSL authentication service (TCPS) as one of the possible options the server can use:

```
SQLNET.AUTHENTICATION_SERVICES = (TCPS)
```

Step 5: Edit listener.ora to create a default TCPS listening port and set the wallet location. The default port number is 2484:

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = ORCL) )
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.222.128)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCPS)(HOST = 192.168.222.128)(PORT = 2484))
    )
  )
WALLET_LOCATION =
  (SOURCE =
  (METHOD = FILE)
  (METHOD_DATA =
(DIRECTORY = /home/oracle10/oracle/product/10.2.0/db_1/wallets/)
 )
)
```

You've completed the setup for the server—it's time to move on to the client.

Step 6: Create an auto-login wallet on the client. Open owm and select Wallet→ New. Enter a password. At this point the wallet is created and is empty. You will create the certificates later. By default the wallet is not created as an auto-login wallet so click on the Wallet, pull down, and click the Auto Login check box. Save the wallet.

Step 7: In listener.ora on the client add the TCPS protocol:

```
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = IPC)(KEY = ORCL))
      (ADDRESS = (PROTOCOL = TCP)(HOST = o11g.dbasecur.com)(PORT = 1521))
      (ADDRESS = (PROTOCOL = TCPS)(HOST = o11g. dbasecur.com)(PORT = 2484))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
  )
```

Step 8: Edit sqlnet.ora on the client and set the wallet location, TCPS authentication, and the SSL_CLIENT_AUTHENTICATION setting:

```
SQLNET.AUTHENTICATION_SERVICES = (TCPS)
SSL_CLIENT_AUTHENTICATION = TRUE
WALLET_LOCATION =
```

```
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
  (DIRECTORY = /home/oracle11/product/11.1/db_1/wallets/)
  )
)
```

Step 9: Edit the server's entry in tnsnames.ora on the client to add a security clause for SSL_SERVER_CERT_DN. This entry tells Oracle to match the server's DN with its host name and makes sure that the server certificate matches the host definitions. This client-side validation helps you ensure that no one is impersonating the server. Edit tnsnames.ora and create an entry that specifies the security property and the TCPS protocol:

```
on0rh4 =
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCPS)(HOST = 192.168.222.128)(PORT = 2484))
  (SECURITY = (SSL_SERVER_CERT_DN="cn=rh4u2x32p,o=dbasecur"))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = on0r4231.guardium.com)
  )
)
```

Now generate the certificates with which the server and client can authenticate each other (and trust each other) using OWM or using orapki as reviewed in Chapter 5. Create the relevant Oracle users that you will connect as identified externally. At that point you can logon to the database by using

```
SQL> connect /@ on0rh4
```

Note that if you chose `SSL_CLIENT_AUTHENTICATION = FALSE` then you will still be connecting over an SSL session in terms of network encryption but authentication will have to be done using the password so you should connect using

```
SQL> connect system@on0rh4
```

Oracle will require you to enter a password.

Two Things to Remember about SSL Authentication using ASO

1. SSL authentication allows you to connect to the database using a certificate rather than a password. It is easier to control certificates and keys than it is to control passwords—so this method is more secure.
2. Setting up SSL also means that the communications will be encrypted—so you get both stronger authentication and network encryption in one configuration.

6.5 HOWTO Configure Kerberos Authentication Using ASO

Kerberos is perhaps the world's most popular authentication method. It was created by the Massachusetts Institute of Technology (MIT) and you can freely download it from MIT at http://web.mit.edu/kerberos/dist/index.html. The protocol was named after the Greek mythological

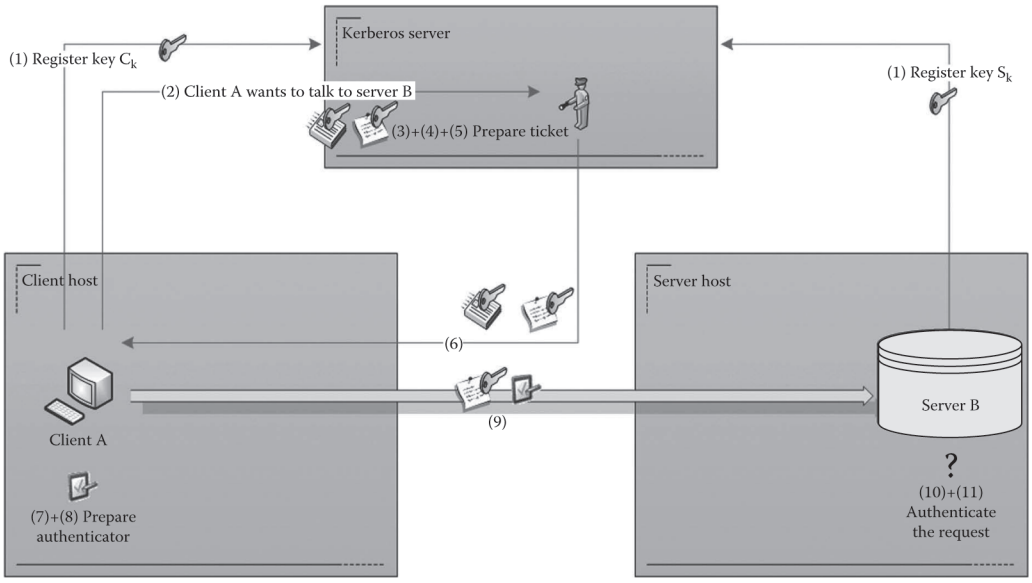

Figure 6.3 Kerberos authentication scheme.

character Cerberus—the three-headed guard dog of Hades. What made Kerberos so popular is, surprisingly, Microsoft. Microsoft fully embraced Kerberos rather than invent a complex authentication method and on any modern Windows environment all authentication occurs using Kerberos as a default.

Figure 6.3 shows how the Kerberos authentication process works (in reality it is more complex and has more steps but for the purpose of this HOWTO this understanding is enough).

1. Kerberos requires an initial setup phase where the parties are registered with the Kerberos server. The Kerberos server is called the Key Distribution Center (KDC) and consists of an Authentication Server and a Ticket Granting Server (TGS). Kerberos uses symmetric keys and does not rely on certificates or PKI. It assumes that there is a trusted third party—the Kerberos server. The KDC maintains a database of secret keys—each entity on the network shares a secret key known only to itself and the KDC. Therefore, in the first step the server and the client need to register with the Kerberos server. At this point, a key for the client C_k and a key for the server S_k are generated. For the client, the secret key is generated by applying a one-way hash function on the client's password.
2. After the client has authenticated itself to the Kerberos server it wants to connect to the database. At this point, the client sends a message to the Kerberos server of the form "client A wants to talk to Server B." This is passed in cleartext to the Kerberos server.
3. The Kerberos server picks a random session key, K.
4. Kerberos encrypts the session key and the string "Server B" with the client key—creating C_k(K, Server B).
5. Kerberos encrypts the session key and the string "Client A" with the server key—creating S_k(K, Client A). This is called the ticket.
6. Both C_k(K, Server B) and S_k(K, Client A) are sent to the client as the response to the request in phase 2.

7. The client knows what the client key is and therefore can decrypt C_k(K, Server B). It extracts the session key K. It also validates that it got the string "Server B"—the server it asked to talk to in the first place.
8. The client uses the session key K as an encryption key to generate K(time, S_k(K, Client A)); it takes the ticket and the current time and encrypts them with the session key. This is called the authenticator.
9. The client sends the ticket and the authenticator to the server.
10. The server has the server key so it can decrypt the ticket—i.e., it can decrypt S_k(K, Client A) and extract the session key K.
11. The server uses the session key to decrypt the authenticator K(time, S_k(K, Client A)). It checks that the ticket within the authenticator is the same as the ticket it received from the client and that the time is correct (i.e., there is no large shift in time that might indicate a replay attack). At this point it knows the client is authenticated and it knows who the client is—Client A was in the ticket itself.

Let's go ahead and see how to set up Kerberos authentication for an Oracle server. Kerberos authentication is an advanced option that is part of Oracle ASO so the first step is to install ASO on your clients and servers. In the sequence below there are three machines—the KDC, the database server, and a client machine. In the example, the name of the machine which hosts the KDC is goose. The database machine runs on saturn and the client connects from a machine called trex. Before you can configure Kerberos authentication for Oracle you have to download and install Kerberos and make sure that ASO is installed for your database. Then, follow these steps to configure Kerberos authentication:

Step 1: Configure a Service Principal (SP) for the Oracle database server.
The format for a SP is kservice/kinstance@REALM. You can choose to use any name for your database but a good convention is to map the service name or SID to kservice and the hostname to kinstance—e.g., orcl/saturn.guardium.com@GUARDIUM.COM. It is very important that the realm be uppercase—if it is not, nothing will work but the errors will not be helpful.
To create the SP start the kadmin utility:

```
[root@goose sbin]# cd /usr/krb5/sbin
[root@goose sbin]# ./kadmin.local
```

Once in kadmin (you'll see `kadmin.local:` as the prompt), add the SP:

```
kadmin.local: addprinc -randkey orcl/saturn.guardium.com@GUARDIUM.COM
```

To verify that the SP has been added:

```
kadmin.local: getprincs
kadmin/admin@GUARDIUM.COM
kadmin/changepw@GUARDIUM.COM
kadmin/history@GUARDIUM.COM
orcl/saturn.guardium.com@GUARDIUM.COM
```

Exit kadmin:

```
kadmin.local: exit
```

If you want to see the activity in the log you can:

```
tail -f /var/krb5/log/krb5kdc.log
```

Step 2: Extract the Service Table from Kerberos. Extract the file from the KDC and then copy the file to the database server (e.g., to /tmp/keytab). On the database server add it using kadmin:

```
kadmin.local: ktadd -k /tmp/keytab orcl/saturn.guardium.com
Entry for principal orcl/saturn.guardium.com with kvno 2, encryption
DES-CBC-CRC added to the keytab WRFILE: 'WRFILE:/tmp/keytab
kadmin.local: exit
```

Run the oklist utility:

```
oklist -k -t /tmp/keytab
```

Step 3: Configure Kerberos authentication. Edit sqlnet.ora and add the following entries:

```
SQLNET.AUTHENTICATION_SERVICES= (BEQ,KERBEROS5)
SQLNET.AUTHENTICATION_KERBEROS5_SERVICE=kservice
```

On the server set the following two initialization parameters:

```
REMOTE_OS_AUTHENT=FALSE
OS_AUTHENT_PREFIX=""
```

You don't have to set the OS_AUTHENT_PREFIX to null—but you should. It is generally a better security practice than leaving OPS$, and more importantly Kerberos names tend to be very long (because of the qualified name and the realm) and you need to remain within the limit of a maximum of 30 characters—no point in wasting three of them on the prefix.

Step 4: Create a Kerberos user. On the KDC run:

```
[root@goose sbin]# cd /usr/krb5/sbin
[root@goose sbin]# ./kadmin.local
kadmin.local: addprinc SCOTTY
Enter password for principal: "SCOTTY@GUARDIUM.COM":
Re-enter password for principal: "SCOTTY@GUARDIUM.COM":
kadmin.local: exit
```

Step 5: Create an externally authenticated Oracle database user:

```
SQL> CONNECT/AS SYSDBA;
SQL> CREATE USER "SCOTTY@GUARDIUM.COM" IDENTIFIED EXTERNALLY;
SQL> GRANT CREATE SESSION TO "SCOTTY@GUARDIUM.COM";
```

Step 6: Get a ticket for the user and then connect with it to the database. Ask the KDC for an initial ticket:

```
trex:oracle10% okinit -f SCOTTY
Password for SCOTTY@GUARDIUM.COM:password
```

Now you can connect to the database without supplying a username and a password. The client will use the Kerberos ticket and the database will authenticate you based on the ticket:

```
trex:oracle10% sqlplus /@orcl
```

If this is the first time you are setting this up, make sure to check that you cannot connect without the ticket. You need to make sure you did not open up the database completely. Destroy the ticket cache using the following command and then try to connect (hopefully unsuccessfully):

```
trex:oracle10% okdstry
trex:oracle10% sqlplus /@orcl
```

Kerberos is one of the most secure authentication schemes. Part of its security stems from the fact that what passes on the network is not enough for an attacker to be able to discover passwords. This is in contrast to the examples with the O5LOGON scheme that you saw in HOWTO 6.1. In O5LOGON, the password hash stored in SPARE4 of the SYS.USER$ table is used. If, for example, someone can access to this table (through some faulty privilege definitions or role assignment) then, with the hashes in hand and through sniffing of the authentication packets, an attacker can get all of the cleartext passwords! This is because all the AUTH_SESSKEY values sent over the wire are encrypted using the hash values. So if you have the password hashes you can decrypt the session keys and then decrypt the password that is eventually passed from the client to the server. To protect O3LOGON or O5LOGON authentication from such an attack you should encrypt the network traffic (see Chapter 7). For example, if you encrypt the communications using ASO network encryption then as soon as the session is established, the encryption begins and the sending of the AUTH_SESSKEY is already encrypted as you can see from Listing 6.2.

Two Things to Remember about Oracle Kerberos Authentication

1. If you have Unix systems, use the MIT Kerberos package. If you have a Windows-only environment, then use the Microsoft domain controller as the KDC.
2. Remember that the realm must be all uppercase.

6.6 HOWTO Configure RADIUS and Two-Factor Authentication Using ASO

The Remote Authentication Dial-In User Service (RADIUS) is also one of the most widely used authentication protocols. It is ubiquitously used to control access to network resources through dialup, Digital Subscriber Line (DSL) providers, or other Internet Service Providers (ISPs). It has also been adopted for its AAA capabilities (authentication, authorization, and accounting) by many networking groups within the enterprise. When you use RADIUS, the client machine requests access to network resources via a Network Access Server (NAS). The NAS issues a RADIUS access request message to the RADIUS server requesting authorization to grant access. This request includes credentials that are provided by the user. The RADIUS protocol does not transmit passwords in cleartext between the NAS and the RADIUS server. A shared secret is used along with the message digests

Without Encryption:

```
0x0000:  E..$..@.@.f.....
0x0010:  .....*.......l.-
0x0020:  ....>........=..
0x0030:  .=...........:.,
0x0040:  .A.............:
0x0050:  ....AA..........
0x0060:  ..............(D
0x0070:  ESCRIPTION=(ADDR
0x0080:  ESS=(PROTOCOL=TC
0x0090:  P)(HOST=o11g.dba
0x00a0:  secur.com)(PORT=
0x00b0:  1521))(CONNECT_D
0x00c0:  ATA=(SERVER=DEDI
0x00d0:  CATED)(SID=on1r4
0x00e0:  531)(CID=(PROGRA
0x00f0:  M=sqlplus)(HOST=
0x0100:  o11g.dbasecur.co
0x0110:  m)(USER=oracle11
0x0120:  ))))

0x0000:  E.....@.@.]B....
0x0010:  .......*.l......
0x0020:  ....?........=..
0x0030:  .=...^..........
0x0040:  ......AUTH_SESSK
0x0050:  EY`...`94C497598
0x0060:  4EE74A3E79C9E85F
0x0070:  DF085B40B64D24C3
0x0080:  A8B4B9E37AA8EF44
0x0090:  82EDA966B44F168C
0x00a0:  C8E4E413BBA8C748
0x00b0:  9210236.........
0x00c0:  AUTH_VFR_DATA...
0x00d0:  ..2B234E4447825A
0x00e0:  570F1D%........A
0x00f0:  UTH_GLOBALLY_UNI
0x0100:  QUE_DBID......44
0x0110:  36399542AB6059FE
0x0120:  1442827C841DD3..
0x0130:  ................
0x0140:  ................
0x0150:  ................
0x0160:  ..........6.....
0x0170:  ...UU...........
0x0180:  ................
0x0190:  ..
```

With Encryption:

```
0x0000:  E..+..@.@.0#....
0x0010:  .....:.......fy[
0x0020:  ............7b%
0x0030:  .7b%.........:.,
0x0040:  .A.............:
0x0050:  ....AA..........
0x0060:  ..............(D
0x0070:  ESCRIPTION=(ADDR
0x0080:  ESS=(PROTOCOL=TC
0x0090:  P)(HOST=abcd32.d
0x00a0:  basecur.com)(POR
0x00b0:  T=1522))(CONNECT
0x00c0:  _DATA=(SERVER=DE
0x00d0:  DICATED)(SERVICE
0x00e0:  _NAME=on1abcd3)(
0x00f0:  CID=(PROGRAM=sql
0x0100:  plus@abcd32)(HOS
0x0110:  T=abcd32)(USER=o
0x0120:  racle11))))

0x0000:  E.....@.@.......
0x0010:  .......:.f{.....
0x0020:  .....}.......7b0
0x0030:  .7b0.d.........N
0x0040:  (&.G..0...B...Y.
0x0050:  #...a.5.....P>..
0x0060:  ..L.....q......9
0x0070:  ?Z!S...5...t!9<-
0x0080:  .).2...<}#....=.
0x0090:  -v..b..$.C.>...h
0x00a0:  ...L...U..A....W
0x00b0:  n..ZW...F.!..$..
0x00c0:  ..C.t..7.‘t.~/.}
0x00d0:  .hu.”....HFbnb..
0x00e0:  ................
0x00f0:  .-^.).”..Gr..G..
0x0100:  u.......t6/...^r
0x0110:  aj......1#.....A
0x0120:  ...Ev8..osyfG/_N
0x0130:  .cG......Zm..rE.
0x0140:  L.....C..N5y....
0x0150:  D...`...e...`T..
0x0160:  .p........t....~
0x0170:  ._1.....].mw..g.
0x0180:  [....u..q{.0..a.
...
```

Listing 6.2 First packets from O5LOGON authentication process—with and without network encryption.

5 (MD5) hashing algorithm. Additionally, the traffic is usually encrypted using IPSEC or some tunneling method to add protection. RADIUS is often used with an Lightweight Directory Access Protocol (LDAP) directory (e.g., Active Directory) where RADIUS performs the authentication but information is stored within the LDAP store.

Oracle databases live within the data center. They may be accessible through a virtual private network (VPN) connection, but this is usually transparent to Oracle—the user logs onto the network over a VPN and then can connect to various resources on the LAN. It is not common to have an Oracle database as a network resource available as a network resource managed by RADIUS. If that is so, why bother learning about RADIUS authentication for Oracle? The reason is that because RADIUS is so pervasive and because it is a fairly simple protocol to implement, it is used by many vendors who offer two-factor authentication as an API through which they can be accessed. If you want to harden your authentication process using two-factor authentication, it is most likely that you will be using RADIUS authentication for your databases.

Let's understand what two-factor authentication means. The most common authentication scheme is the one which is based on a password—i.e., based on "something you know." This is considered by many to be a weak scheme—passwords can be guessed, can be seen when you type them in, and can be shared by multiple users. Two-factor authentication makes the logon dependent on two things—usually on "something you know" and "something you have." Almost all two-factor authentication schemes still use passwords but they augment that with one more thing—usually some physical device that you must have to logon. This physical device can be a USB key, can be a card that you carry with you, etc.—the important thing is that it is something that you have, and only you can have. There is only one device in the world that could be used to logon to your account.

Many companies make two-factor authentication devices and there are many forms that it can take. Figure 6.4 shows three such examples. On the left is an RSA SecureID token. You carry this little token with you (e.g., on your keychain). As you can see the token presents a number with eight digits. This number changes every 60 s. Each such token has a unique internal key so every

Figure 6.4 Example of two-factor authentication devices.

such token will switch numbers in a different sequence that cannot be guessed. Therefore, if you have 30,000 employees in your organization and everyone has such a token, the number sequences are all unrelated. However, they are NOT random. In fact, a centralized RSA ACE authentication server knows at any point of time what number each token is showing. When you log onto a system that delegates authentication to the ACE/Server, you provide the number on your token in addition to your password. Because the ACE/Server can compute the number sequences on each device it can verify that you not only know the password but that you indeed have the physical device.

The two tokens on the right-hand side of Figure 6.4 use a different scheme. In this scheme when you want to be authenticated, the server will give you a random number. You then key in that random number on the keypad. The device performs some mathematical transformation on this number using a key embedded into the device and shows you the output on the liquid-crystal display (LCD). You then enter this number as part of the authentication process. Because the authentication server knows the keys per device, it too can compute the transformation and verify that you indeed have the device.

There are two scenarios that can occur when you use two-factor authentication. Figure 6.5 shows the basic scenario. Let's stick with the RSA SecureID token as an example. In this case, when you log onto Oracle you provide your username, your password, and the number you see on the token (step 1 in Figure 6.5). Oracle forwards this information to the authentication server using a RADIUS protocol (step 2). The authentication server responds to the Oracle server (step 3) and Oracle responds to the client (step 4); you are now logged on to Oracle.

As mentioned, the number on the SecureID device changes every 60 s. What happens if you type in the value you see on the device 2 seconds before it changes and then try to logon? By the time you key it in and press enter, 2 seconds may pass and when the authentication server performs its check, it will have a different number—in which case the numbers will not match. In this scenario, shown in Figure 6.6, the authentication server will initiate a challenge/response sequence.

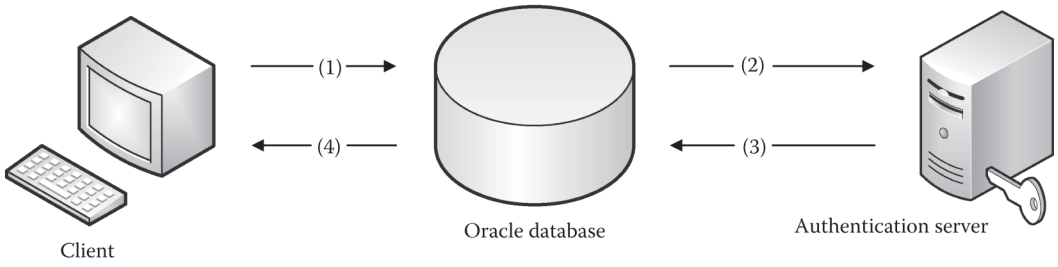

Figure 6.5 Synchronous two-factor authentication sequence.

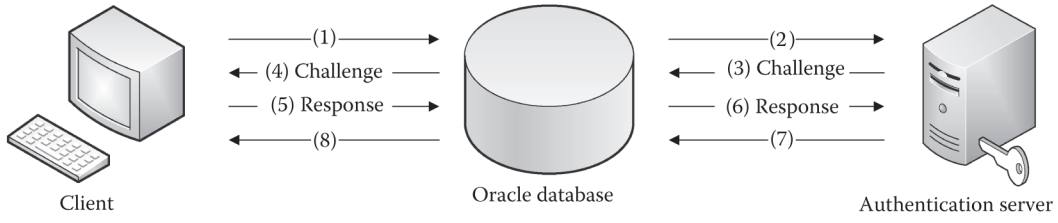

Figure 6.6 Challenge/Responses in two-factor authentication sequence.

In this scenario the authentication server will not immediately reject the logon. Instead, it sends a challenge as shown in steps 3 and 4. You will see the challenge because your logon screen will have some form of message that tells you to wait a full cycle until the number changes once more and then logon using the new number. This time you have 60 seconds until the next change and therefore the response (steps 5 and 6) will likely succeed. To try to avoid this complexity most devices such as SecureID show you how long you have left until the next change. In Figure 6.4 the bars on the left of the number show you how much time is left until the next change. When the number changes (and you have 60 seconds until the next switch) there are six bars. One bar is dropped every 10 seconds so you can always tell when the number will switch next. If you see that you have only one bar left, you will usually wait until the number switches again before attempting a logon. With the devices with the keypad shown on the right of Figure 6.4, there is always a challenge/response sequence—the authentication server issues a random number that you type in using the keypad (the challenge) and the resulting number is your response.

Let's move on to setting up RADIUS authentication using ASO. Setting up RADIUS authentication for Oracle looks the same regardless of which system you are using for authentication. Sometimes the challenge/response phase requires custom handling—but for the most part, the fact that the RADIUS protocol is a standard supported by ASO allows you to interface Oracle with many systems. In this specific example you'll follow the procedure for integrating Oracle with the RSA ACE/Server using RADIUS. The sequence for this setup is

Step 1: On your ACE/Server start the RSA Authentication Manager application. Click on Agent Host→ Add Agent Host. You need to register your database server. Put in the host name, address, and select Net OS Agent as the Agent type as shown in Figure 6.7. Uncheck Requires Name Lock and all the other default options as shown in Figure 6.7.

Step 2: Click on Secondary Nodes and enter all the hostnames/IP addresses for Oracle.

Step 3: Click on Create Node Secret File to generate the node secret file.

Step 4: Using the import utility, convert the .rec file to a .key file. Save this key on the Oracle server in $ORACLE_HOME/network/security/radius.key and ensure that the file is protected through file permissions.

Step 5: On the database server and the clients, edit sqlnet.ora and add the RADIUS authentication method:

```
SQLNET.AUTHENTICATION_SERVICES= (BEQ, RADIUS)
```

Step 6: Start up netmgr on the database server. Select Oracle Advanced Security and go to the Other Params tab. Select RADIUS from the pull down and enter the RADIUS server properties as shown in Figure 6.8.

Step 7: Set the following two initialization parameters:

```
REMOTE_OS_AUTHENT=FALSE
OS_AUTHENT_PREFIX=""
```

Step 8: Create a database user identified externally:

```
SQL> create user ronb identified externally;
SQL> grant create session to user ronb;
```

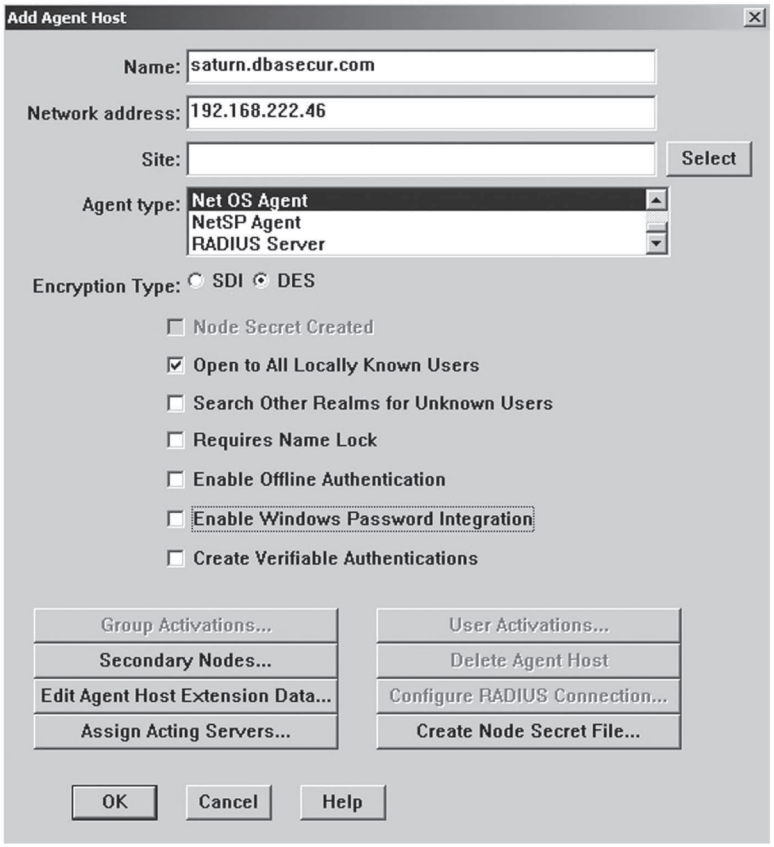

Figure 6.7 Adding the database as a RADIUS client.

When you connect using this user the database will authenticate it through the RSA/ACE server using a RADIUS protocol.

Two Things to Remember about Oracle RADIUS Authentication

1. Use RADIUS authentication when you want to add two-factor authentication to the database.
2. Make sure to use a package that has a well-documented setup procedure for the challenge-response sequence.

6.7 Discussion: Protect Your Password Hashes

As you saw at the end of HOWTO 6.5 you must protect your password hashes in USER$ through privileges and through auditing. With the password hashes, an attacker can get access to the plaintext passwords if they can sniff the network and you don't encrypt the network traffic. This type

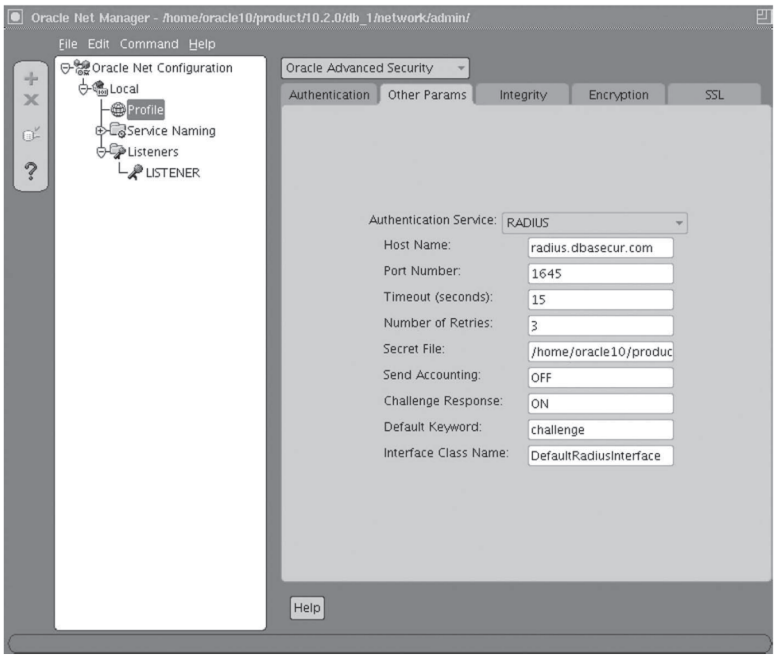

Figure 6.8 Setting RADIUS settings using netmgr.

of attack relies on the fact that the session keys are encrypted using the password hashes. With the password hashes, an attacker can know the session keys and thus the plaintext passwords. Password hashes can also be used in a password cracking attack. Because the password hashing algorithm is well known, an attacker can run a program that guesses passwords, computes the hash value, and compare it with your values. They can do this on their own database or even in a C program optimized for fast iterations. If you choose your passwords randomly, this may take a very long time but usually passwords are not random and a brute force attack can be effective. For example, one of the better programs, orabf, calculates over 1 million hashes per second. For passwords of length eight this can take a few days to complete—that's pretty scary when you think about it.

These are only two examples in which password hashes can be used to take over a database—and there are many more. The bottom line is that you need to think of password hashes as highly sensitive data that must be secured well. Alternatively, you may decide that this is an area that you don't want to own and you would prefer to delegate this—i.e., not use the database for authentication, not keep passwords within the database, and rely on an external authentication system as described in this chapter.

When you use database authentication you must secure your password hashes as though they were the passwords themselves. At a minimum you must do the following:

1. Use access control to ensure that access to the hashes (PASSWORD in SYS.USER$ and DBA_USERS prior to 11g and SPARE4 in SYS.USER$ in 11g) is very limited.
2. If you use an advanced solution such as Oracle Database Vault then you can place better controls on this data (see Chapter 17).
3. Periodic audits to review privileges through which users may access these columns.
4. Continuous auditing and real-time alerting on any access to this data by any user.

Database security

HOWTO Secure and Audit Oracle 10g and 11g

Ron Ben Natan

Foreword by Pete Finnigan

Oracle is the number one database engine in use today. The fact that it is the choice of military organizations and agencies around the world is part of the company's legacy and is evident in the product. Oracle has more security-related functions, products, and tools than almost any other database engine. Unfortunately, the fact that these capabilities exist does not mean that they are used correctly or even used at all. In fact, most users are familiar with less than 20 percent of the security mechanisms within Oracle.

Written by Ron Ben Natan, one of the most respected and knowledgeable database security experts in the world, ***HOWTO Secure and Audit Oracle 10g and 11g*** shows readers how to navigate the options, select the right tools and avoid common pitfalls. The text is structured as *HOWTOs* — addressing each security function in the context of Oracle 11g and Oracle 10g.

Among a long list of *HOWTOs*, readers will learn to —

- Choose configuration settings that make it harder to gain unauthorized access
- Understand when and how to encrypt data-at-rest and data-in-transit and how to implement strong authentication
- Use and manage audit trails, and advanced techniques for auditing
- Assess risks that may exist and determine how to address them
- Make use of advanced tools and options such as Advanced Security Options, Virtual Private Database, Audit Vault, and Database Vault

The text also provides an overview of cryptography, covering encryption and digital signatures and shows readers how Oracle Wallet Manager and orapki can be used to generate and manage certificates and other secrets.

While the book's 17 chapters follow a logical order of implementation, each *HOWTO* can be referenced independently to meet a user's immediate needs. Providing authoritative and succinct instructions highlighted by examples, this ultimate guide to security best practices for Oracle bridges the gap between those who install and configure security features and those who secure and audit them.

CRC Press
Taylor & Francis Group
an **informa** business
www.taylorandfrancisgroup.com

6000 Broken Sound Parkway, NW
Suite 300, Boca Raton, FL 33487
270 Madison Avenue
New York, NY 10016
2 Park Square, Milton Park
Abingdon, Oxon OX14 4RN, UK

AU4127
ISBN: 978-1-4200-8412-2
90000
9 781420 084122

www.auerbach-publications.com