

A Simple Oracle Host-Based Scanner

Pete Finnigan 2001-12-13

A Simple Oracle Host-Based Scanner

by *Pete Finnigan*

last updated December 13, 2001

Introduction

As with any large software package, the default installation of Oracle does not provide for the most secure system out of the box. Indeed, some aspects of the default installation are remarkably insecure. There is a high dependency on the database administrator (*dba*) to ensure that the system is correctly configured, thereby avoiding some of these issues.

This paper will explore the scanning of an installation of Oracle's RDBMS and, in doing so, will investigate some common security deficiencies. A short paper cannot possibly cover all known security weaknesses in an Oracle installation, so only a small sample of the common issues will be covered. The paper is written around a simple script that the reader can download from www.pentest-limited.com/scanner.sql. This paper does not attempt to replace a complete Oracle security audit or specific Oracle penetration test; furthermore, it is debatable whether the script on which this article is based can be described as a scanner. The intention is to show how relatively easy it is to check for some common, simple installation vulnerabilities that can cause security problems. The script has been written using Oracles standard internal language PL/SQL to assist with portability. For the purposes of this paper, the script is restricted to the RDBMS and covers a limited selection of tests.

The Checks

As this paper discusses a script and the simple tests that it performs, it is structured around each of these tests. Each test checks for instances of the topics described in each section below. In case the readers missed it in the previous paragraph, the script is available for download from www.pentest-limited.com/scanner.sql.

Default Users - Check for Known Passwords

The easiest way to access the database is by utilizing one of the many default accounts. There are a large number of these accounts with known default passwords. These default accounts will not be installed in most databases, but a few key ones will be along with any that users have installed as part of examples or additional features.

The first part of this script hard codes all of the default usernames, passwords and their hashes, as stored in the database. The check simply compares the hash stored in the database with the one provided in this script, if there is a match then the default users password has been left as the default.

Clearly the solution to this issue is to change the password or delete the user if it is not needed. If the password of some of the default users is changed other considerations need to be made. The user "dbsnmp" for instance is used by the intelligent agent to perform tasks from a remote console as part of the Oracle Enterprise Manager (OEM). If the password is changed and OEM is being used, then the password also needs to be changed in the file `$ORACLE_HOME/network/admin/snmp_rw.ora` file.

Check for User Passwords that are Easy to Guess

This check is not included in the script provided, as doing so would detract from the script's simplicity. The depths that the reader can go to checking normal users' passwords are up to the reader. The simplest check is to see if the user has set his / her password to the same as the username, you could go further and use a simple dictionary and check each user against that. It is relatively simple to select all of the usernames from the view `ALL_USERS` and then generate another `SQL` script to check. An example could be:

```
SQL> set head off
SQL> set feed off
SQL> set verify off
SQL> set termout off
SQL> set pages 0
SQL> spool check.sql
SQL> select 'connect ' || username || '/' || username || ';'
      2 from all_users;
```

```
connect SYS/SYS;
connect SYSTEM/SYSTEM;
connect OUTLN/OUTLN;
connect DBSNMP/DBSNMP;
connect MTSSYS/MTSSYS;
connect PETE/PETE;
connect RAJK/RAJK;
SQL> spool off
```

Then run the output script to check if any users have a password that is the same as their username. Other "check" scripts can be generated in a similar way using a dictionary either read in from a file or held in a database table.

Connect as INTERNAL

The alias "INTERNAL" has been deprecated in Oracle 8i and removed altogether in 9i. The alternate syntax `connect sys/sys as sysdba` is used instead. For Oracle 8i the user alias "INTERNAL" has a default password of "ORACLE". If the user "SYS" has not had its password changed, then trying to connect as internal will work using the password "oracle".

Check for Users Who Have the Privilege "DBA"

This check is also not included in the script, purely for reasons of brevity. There are now a lot of system privileges in Oracle8i and default roles with powerful privileges. All of the system privileges that are available can be seen by running the following select statement as the user "SYS" or by a user that has access to this object and pre-pending the username to the select.

```
SQL> select * from system_privilege_map;
```

An attacker with malicious intent can use any number of system privileges. I check for two of the system privileges in this script later. It could also be useful to check regularly which users have been granted the role "dba". Of course, this will not protect against a specific user abusing a specific privilege; but, out of laziness, a hacker could grant himself "dba" or use an account with this privilege, as it is easy to find. The following SQL shows how:

```
SQL> select grantee
       2  from dba_role_privs
       3  where granted_role='DBA';
```

```
GRANTEE
```

```
-----
```

```
CTXSYS
```

```
SYS
```

```
SYSTEM
```

```
SQL>
```

Don't forget that roles can be granted to roles so to find these users, the check should be hierarchical.

Check For Users and Roles that Have Privileges With "ANY" Included

If a user or a role has privileges that include the word "ANY" in them, this can be useful to an attacker. A good example would be the privilege "SELECT ANY TABLE". An attacker may wish to see the data in another user's table, but may not have access to that user or be able to see its table. The table may be private and not exposed to any other user for access. But if the attacker can find another user that

has the "SELECT ANY TABLE" privilege, then he can still read what he wants. Here is an example:

```
SQL> connect dbsnmp/dbsnmp
Connected.
SQL> select name,password from sys.user$;
select name,password from sys.user$
                                *
ERROR at line 1:
ORA-01031: insufficient privileges
```

```
SQL> connect sys/change_on_install
Connected.
SQL> grant select any table to dbsnmp;
```

Grant succeeded.

```
SQL> connect dbsnmp/dbsnmp
Connected.
SQL> select name,password from sys.user$;
```

NAME	PASSWORD
SYS	D4C5016086B2DC6A
SYSTEM	D4DF7931AB130E37

2 rows selected.

```
SQL>
```

In this example I have shown that the user "dbsnmp" cannot see the "sys"-owned "user\$" table, but after I grant "SELECT ANY TABLE" to this user, this table can then be selected from.

Check For Users That Have "with admin" or "with grant"

These are two rights that can be added when privileges on objects are granted. Object privileges are granted to users or roles to allow them to access or alter an object. Adding the "with grant" option means that the user who has been granted the privilege can then grant the privilege on to another user. This means that a user can leave the management of certain objects to the user who has been granted the privilege without giving access to their entire schema.

The second right that can be granted with system privileges is "with admin". Object privileges allow a user to alter data, system privileges allow the user to change the schema itself and to create and alter objects. The "with admin" is similar to "with grant" and basically allows the receiver of a system

privilege that has "with Admin" to grant the privilege on to another user.

This part of the script prints out any granted privileges with either of these rights. These privileges, and the users that have been granted them, can provide an easy way for an attacker to find a way to get at objects or data, if he can guess an easy users' password without having to gain access as oracle or a "dba" or any higher level user.

Check External Users

External users allow an application user to log in through the operating system and let it authenticate access to the database itself. This feature can be useful if you want to run batch jobs or cron jobs and do not want to hard code usernames and passwords in scripts or pass them on the command line so that they could be grabbed using an operating system command such as `ps -ef | grep sqlplus`.

This part of the script identifies some of the database accounts that are managed by the operating system. These have passwords in the `SYS.USER$` shown as `EXTERNAL` to identify that they are O/S-authenticated. While using these accounts to prevent passwords being visible on the command line or being hard coded in scripts is a valid approach to system security, an attacker finding an external user and gaining access to it will also gain access to the database. So users should beware and consider not using external accounts.

Besides this one, there are a number of ways of using and identifying users who are authenticated via the operating system. Looking for `OPS$` prepending a user name, as it could indicate external users. Some initialization file parameters are also used to control external users, these are: `remote_os_authent` and `os_authent_prefix`. It is beyond the scope of this paper to review these parameters and other forms of external authentication. It is sufficient here to find some of the external users and to note that an attacker could use them maliciously.

System Privileges - ALTER SYSTEM and CREATE LIBRARY

There are many system privileges that are available in an Oracle database. These privileges can be granted to a user directly or to a role. Selecting them from a system view, one can see the privileges available. This view can be generated with the following SQL statement.

```
SQL> select * from system_privilege_map;
```

PRIVILEGE	NAME	PROPERTY
-3	ALTER SYSTEM	0
-4	AUDIT SYSTEM	0

```

-5 CREATE SESSION                                0
-6 ALTER SESSION                                0
...
-225 ALTER ANY OUTLINE                          0
-226 DROP ANY OUTLINE                           0
-227 ADMINISTER RESOURCE MANAGER                 1
-228 ADMINISTER DATABASE TRIGGER                 0

```

126 rows selected.

SQL>

To view which system privileges are granted to a role or a user, use the following SQL statement.

```
SQL> select * from dba_sys_privs;
```

GRANTEE	PRIVILEGE	ADM
-----	-----	----
AQ_ADMINISTRATOR_ROLE	DEQUEUE ANY QUEUE	YES
AQ_ADMINISTRATOR_ROLE	ENQUEUE ANY QUEUE	YES
...		
CONNECT	CREATE TABLE	NO
CONNECT	CREATE VIEW	NO
CTXSYS	ADMINISTER DATABASE TRIGGER	NO
CTXSYS	ADMINISTER RESOURCE MANAGER	NO
CTXSYS	ALTER ANY CLUSTER	NO
...		
TIMESERIES_DEVELOPER	CREATE VIEW	NO

505 rows selected.

SQL>

I have chosen to highlight just two system privileges in this paper as examples. These are "ALTER SYSTEM" and "CREATE LIBRARY". The first privilege could allow an attacker to change system parameters and more. The second privilege, "CREATE LIBRARY", allows the user to create a library. This privilege is very dangerous as it may make possible an escalation of privileges.

utl_file_dir Location

There are a lot of initialisation parameters used by an Oracle database, quite a few of which can be construed as being dangerous from a security perspective. The parameter controlling the directory that the database package UTL_FILE reads and writes to is a good example. If this parameter is set to "" (don't include the quotes) then the package UTL_FILE can be used to read or write to any directory

or file on the machine where the owner of the Oracle installation has access (usually the user "ORACLE"). It is possible for a user of the database with the privilege "ALTER SESSION" to dump the library cache (see http://www.pentest-limited.com/utl_file.htm for an example) to a trace file and then read anything from this file with `UTL_FILE`. If any users have been added or passwords changed, then the clear text password can still be viewed (so long as it has not been aged out of the cache).

The next check is to see where all of the `dest` parameters are pointing by listing them out (this is just for information). The last check in this section checks the main trace directory: if `utl_file_dir` is the same as `user_dump_dest` then you would have similar problems to those described above. It is also worth noting where `utl_file_dir` points (if neither of the above cases are satisfied), as someone could have pointed it at a directory that you don't people to read.

Database Links That Have Clear Text Passwords

Database links allow one to access data in another database easily. These links can even be hidden using synonyms, so that the `SQL` does not hint at the location of the actual data. A sample database link could be created as follows:

```
Create database link test_link
Connect to scott identified by tiger
Using 'oids' ;
```

The "using" clause supports either a complete connect string or the service name as defined in the `tnsnames.ora` file. To retrieve data from the other database you could use the following example:

```
SQL> select * from scott.emp@test_link ;
```

There are two types of database links: public and private. Public database links allow any user who includes PL/SQL code to access the data in the remote database through the database link. Private database links offer better security, as the remote data is only accessible if you, or any PL/SQL code, are connected as the account that owns the link.

It is also possible to create the database link to connect to the remote database as the connected user rather than a dedicated user. The test in the sample script allows you to view the username and passwords assigned to any database links in the database, as well as the authorized user and password.

Database links are a good way for a hacker to access a secure database from an insecure one. The attacker can access a weak database and find out the username and password of a user in a secure one. He can then access the data using the link or log in directly.

Conclusions and Other Information

Other information can be gleaned from the output of this script or it can be easily changed to provide other details not covered. It should be relatively easy for someone to use the sample script as a starting point, adding to or modifying the script to cover other site specific issues. The script should allow additional checks to be added to the script so that regular reviews of the database can be performed automatically.

One such check might be to see how many users have the role "dba". This is a dangerous role if the password for any of the users with it is compromised. It is important to review what each user can do in your database and also which objects are available for execution or access and by whom.

It is also useful to monitor all of the known exploits and holes found in Oracle databases and protect against them. It is important to monitor and secure the O/S that the database runs on and the network to which it is connected. It is worth checking the status of objects and the PL/SQL code in the database to make sure no user has altered anything or added objects for future use.

Oracle also provides many packages and objects that have execute privileges granted to public, which is to say that every user can access them. An attacker can use some packages such as UTL_FILE, UTL_HTTP and UTL_TCP maliciously. The point here is that many of the public packages are accessible and probably shouldn't be. Review what access is granted and revoke everything that is not needed!!

Once again, the script for the simple Oracle scanner to review some basic aspects of an Oracle installation can be downloaded from <http://www.pentest-limited.com/scanner.sql>. The author invites readers to e-mail him with any comments, questions or suggestions regarding the script.

Peter Finnigan is Director of Application Development at PenTest Limited, which provides products and services to clients with data of a sensitive nature. Prior to his appointment at PenTest Limited, Peter was involved in developing new financial applications based around the Oracle product set. He is currently ranked amongst the world's top five experts in Oracle security and is due to complete his book on this subject late next year.

[Privacy Statement](#)

Copyright 2006, SecurityFocus