Some thoughts on Oracle passwords

By Pete Finnigan

The Problem

Most companies have trouble with users who tend to use the same password for all of their accounts on applications and systems used or even worse use the username as their password. This makes it very easy for an attacker to guess passwords or for some malicious internal user to log on as an unsuspecting colleague and steal data or do some damage.

Its up to the reader to resolve the above issues but I can offer some help with setting stronger passwords. The following sections offer some tips and information:

- *Some facts about Oracle passwords*

  Most people who use Oracle and create users or change passwords are familiar with the rules for what characters can be used for a password. Most people know that the password has to start with a letter and can be up to 30 characters long and can after the first letter be any letter, digit or _ or # or $. The result is also not case sensitive as we can see below

```
  SQL> alter user pete identified by pete;

  User altered.

  SQL> select username,password from dba_users where
username='PETE';

  USERNAME                       PASSWORD
  ------------------------------ ------------------------------
  PETE                           4040619819A9C76E

  SQL> alter user pete identified by PETE;

  User altered.

  SQL> select username,password from dba_users where
username='PETE';

  USERNAME                       PASSWORD
  ------------------------------ ------------------------------
  PETE                           4040619819A9C76E

  SQL>
```

  Quite clearly this reduces the possible number of combinations of password quite a bit if someone were to brute force an account by attempting multiple logins.

What is not usually realised or remembered is that Oracle passwords can actually contain anything as long as the password is encased in double quotes. This is a feature that the *dba* and security managers should be aware of and should enforce users to use. The case insensitivity is not removed:

```
  SQL> alter user pete identified by pete;

  User altered.

  SQL> select username,password from dba_users where
username='PETE';

  USERNAME                       PASSWORD
  ------------------------------ ------------------------------
  PETE                           4040619819A9C76E

  SQL> alter user pete identified by "Pete";

  User altered.

  SQL> select username,password from dba_users where
username='PETE';

  USERNAME                       PASSWORD
  ------------------------------ ------------------------------
  PETE                           4040619819A9C76E

  SQL> alter user pete identified by "PeTe";

  User altered.

  SQL> select username,password from dba_users where
username='PETE';

  USERNAME                       PASSWORD
  ------------------------------ ------------------------------
  PETE                           4040619819A9C76E

  SQL>
```

We can however use any character we choose!

- *Create passwords with CTRL characters*

  One useful feature of this is to set a password with control characters in it. This will then make it difficult for a casual hacker to use the password interactively say from SQL*Plus. They would have to know that the password was set up in this way and prepare a file with the username/password in it with control characters in place. This method can also be useful with database links where the password is visible by anyone querying `SYS.LINK$`. It would not prevent a determined attacker but this feature could prevent casual abuse.

```
  SQL> create user ch_chk identified by "ch^Mchk";

  User created.
```

```
SQL> select username,password from dba_users where
username='CH_CHK';

USERNAME                       PASSWORD
------------------------------ ------------------------------
CH_CHK                         8FD0566E5320AA31

SQL> connect ch
ERROR:
ORA-01017: invalid username/password; logon denied


Warning: You are no longer connected to ORACLE.
SQL>
```

As you can see I was able to create a user with a password with a carriage return in the middle of it and I was not able to log in with it on the command line.

- *Lock an account the old way*

  If there are many user accounts in your Oracle database it is well worth monitoring their use and removing or disabling accounts that are no longer needed.

  If you wanted to lock out a users account before Oracle 8 then you needed to revoke the role that gave the privilege `CREATE SESSION` from the user you wished to lock out or revoke `CREATE SESSION` explicitly from the individual users. This privilege is usually included in the role `CONNECT`.

```
SQL> connect pete/pete
Connected.
SQL> connect sys/change_on_install
Connected.
SQL> revoke connect from pete;

Revoke succeeded.

SQL> connect pete/pete
ERROR:
ORA-01045: user PETE lacks CREATE SESSION privilege; logon
denied


Warning: You are no longer connected to ORACLE.
SQL>
```

  There is a second way to lock out an account pre Oracle8 and that is to use the undocumented feature that allows a password to be inserted directly into the dictionary table `SYS.USER$` and avoid the username/password encryption routines. This feature is used in the import routines to allow a user to be imported into a database with its original password. The password hashing routine uses a modified DES algorithm (see Oracle Security Handbook by Oracle press) and stores a 16 bit hexadecimal hash in the `SYS.USER$ PASSWORD` column. So by using the same undocumented feature its possible to add an impossible password to

the database for a particular user thereby ensuring that a password supplied by anyone, attackers included, cannot be encrypted to this value. An example is shown here:

```
SQL> connect pete/pete
Connected.
SQL> alter user pete identified by values '!impossible!';

User altered.

SQL> connect pete/!impossible!
ERROR:
ORA-01017: invalid username/password; logon denied


SQL>
```

As you can see by specifying a password hash that is not a valid hex number the hash is an impossible one to get from any password supplied.

- *Oracle 8i User management features*

  With oracle8 there are now new password / user management features added. It is now possible to lock out a user explicitly as the following example shows:

```
SQL> connect pete/pete
Connected.
SQL> connect sys/change_on_install
Connected.
SQL> alter user pete account lock;

User altered.

SQL> connect pete/pete
ERROR:
ORA-28000: the account is locked


Warning: You are no longer connected to ORACLE.
SQL> connect sys/change_on_install
Connected.
SQL> alter user pete account unlock;

User altered.

SQL> connect pete/pete
Connected.
SQL> spool off
```

It is possible to set the parameter `PASSWORD_LOCK_TIME` so that the users account will unlock after the specified time period. If this parameter is set to `UNLIMITED` it will never unlock. This feature can be useful if a user fails to log in correctly after a specified number of attempts set by the parameter `FAILED_LOGIN_ATTEMPTS` and his account is locked out.

You can also force a user to change their password by issuing the following command:

```
SQL> connect pete/pete
Connected.
SQL> connect sys/change_on_install
Connected.
SQL> alter user pete password expire;

User altered.

SQL> connect pete/pete
ERROR:
ORA-28001: the password has expired


Changing password for pete
Password changed
Connected.
SQL> alter user pete identified by pete;

User altered.

SQL>
```

As you can see at the end of this script there is a serious flaw with the password management features. That is you can bypass them very easily and just use old syntax of `ALTER USER`. This will also be shown further down when we look at the password strength function.

There are many more features to the new password management including forcing a user to change his password or setting the lifetime of the password or the reuse time. All of these features are enabled by running the script `$ORACLE_HOME/rdbms/admin/utlpwdmg.sql`. Beware though that once you have turned on these features its is very difficult to turn them off as there is no un-install script available.

- *Oracle 8 password command*

  Oracle 8 provides a command to change a users password called `PASSWORD`. Using this command allows the password strength features to be used. This is a feature of SQL*Plus and as such can only be used in that tool so the original `ALTER USER` syntax still works and can be used to bypass this functionality. An example of its use is shown below.

```
SQL> connect pete/pete
Connected.
SQL> password
Changing password for PETE
Password changed
SQL> connect sys/change_on_install
Connected.
SQL> password pete
Changing password for pete
Password changed
SQL>
```

As you can see when changing a password for the user you are logged in as the old password needs to be supplied but if changing the password for another user if you are a *dba* this is not the case. The password verify function described next is called when this feature is used and it can be seen that a malicious user if he got the right access could replace the verify function with a trojan to collect peoples old and new passwords.

- *Oracle Password strength function*

  This feature was also added in Oracle8 and a sample function is installed when the script `$ORACLE_HOME/rdbms/admin/utlpwdmg.sql` is run. It provides a sample function called `verify_function`. It provides basic rules such as the password must not be the same as the username and at least 4 characters long and contain at least one digit, one punctuation mark and at least one alphabetic character. It also checks that a new password is different from the previous one by at least 3 characters.

  You can of course modify the function provided by Oracle or write your own completely to provide your own corporate rules. The function does not need to be called `verify_function` and must be owned by *SYS*. The function is then assigned to a specific users or the default system profile. Hence it is possible to use different functions for different profiles. So you can control different password strength policies for different departments.

  It would be installed by adding it to a profile as follows:

```
alter profile default limit
password_verify_function verify_function;
```

  Adding things like failed login attempts is done in the same manner

```
alter profile default limit
failed_login_attempts 3;
```

- *Audit user activity*

  Oracle auditing is a powerful feature and should be used sparingly. The amount of output generated can be quite large and far too much auditing could start to affect performance. It is also important to move the audit table `SYS.AUD$` from the `SYSTEM` tablespace if auditing is set up to use the database as the database would hang if all the free space in the `SYSTEM` tablespace were to be used up by audit records. This could be a potential source for a remote DOS if someone knew there was login audit turned on and they kept trying to login with a known incorrect password and failed login attempts are turned off.

  That said its worth auditing user login attempts for both successful and unsuccessful attempts. This would highlight potential hacker attempts to guess passwords, it might highlight account abuse internally within an organisation and it could be used to identify accounts that are not in use so that they could be locked.

Auditing is quite easy; the user needs to be granted the system privilege `AUDIT SYSTEM`. The initialisation file parameter `audit_trail` should be set to either `NONE`, `OS` or `DB`. If you set it to the Operating system then the parameter `audit_file_dest` should also be set to a suitable directory.

Auditing your users connection attempts can be done with the following commands:

```
SQL> audit connect whenever unsuccessful;
SQL> audit disconnect whenever unsuccessful;
```

You can limit the audit to specific users and also you can audit on success as well. The results of the audit trail if set to `OS` are stored in audit files pointed to by the parameter above. If it's set to `DB` then you can select the results from the table `SYS.AUD$`. To watch for specific abuses you would write custom reports with `SQL` or create views on the audit table.