Chapter 7

Passwords and Password Controls

Solutions in this chapter:

- **Configuring Strong Passwords**
- **Password Controls Using Oracle Profiles**
- **OS Authentication**
- **Automated Scanning for Weak Passwords**

- ☑ **Summary**
- ☑ **Solutions Fast Track**
- ☑ **Frequently Asked Questions**

Introduction

Authentication is the gateway to the database, and for the vast majority of Oracle systems, the gatekeeper requires no more than a valid username and password pair to allow anyone to pass. One can argue over the merits of username- and password-based authentication, and can make claims about external authentication mechanisms being better, stronger, faster. However, those arguments will not be made here. Instead, we're going to focus on what Oracle has given us, and what we see practically every production Oracle system using. The native Oracle authentication mechanisms are secure enough for almost all systems, but only when used properly.

Passwords are a critical component of your Oracle security infrastructure. This chapter focuses on establishing a system that ensures that all your passwords are difficult to guess, then configuring protections to thwart password-guessing attacks against your databases.

Configuring Strong Passwords

Keeping unauthorized individuals out of your Oracle databases requires you to ensure that every user has a strong password. Passwords are important. They hold the key to each database, allowing anybody with the right password into the system. Passwords are also a target of attackers and their powerful automated attack tools. There are more password crackers out there than any other kind of hacker tool. Try searching in Google. I got 1.7 million results when I searched on the term "password cracker". The last result on the first page (see Figure 7.1) was particularly interesting.

Figure 7.1 Oracle Password Cracker

You don't make it to the front page on a list of nearly 2 million search entries without a lot of clicks. The notorious John the Ripper was just a few entries above this one. The point is that it takes little more than the ability to point and click to download a powerful password-cracking tool. It's not much more difficult to point that tool at a database and start breaking in. These tools are out there and a large

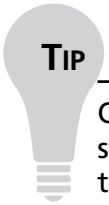
number of people are using them. Strong passwords are the first line of defense against these attack tools.

What Makes a Password Weak?

Weak passwords are easy to guess. This includes more than the passwords that are easy for a person to guess, but also those that are easy for a computer to guess. Password crackers are computer programs that are built to guess passwords. Password crackers can work in different ways, but the most common is dictionary-driven, where the tool cycles through a dictionary of passwords, trying each password in the dictionary for each known account (or even every username in a separate dictionary) until it is able to log in. Simply put, if a password is likely to end up in a password cracking tool's dictionary file, then it is a weak password. But how can you tell?

Start with the English dictionary. If a word is in there, it's easy to guess. Next add in cities and sports teams. Add numbers to make up dates, like birthdays or anniversaries. Finally, add simple patterns like 12345 or qwerty. You will find most if not all of these in a typical password cracker dictionary file.

Usernames are also weak passwords. It's very common to see accounts in Oracle databases where the password is the same as the username. This should really be no surprise, if anything this is a trend that Oracle themselves started. The majority of the accounts Oracle includes in the database by default have their password set to their username.



TIP

Oracle takes steps to protect passwords in the system. First, all passwords are stored as a password hash, never in cleartext. Looking at the password hash tells you nothing about the password. Second, Oracle blocks access to the password hashes, storing them in the SYS schema and only displaying them in the database administrator (DBA) views (in Oracle 11g, even the DBA_USERS view does not show passwords).

Usernames, however, are not protected. Anyone with access to the database can get a list of users by selecting from the ALL_USERS view (SELECT granted to PUBLIC by default). This makes it easy to test every account in the database for a password that equals the username, and potentially gain unauthorized access to the system.

Another form of password cracking is called brute-force password guessing. Brute-force is more aggressive than a dictionary attack, primarily focusing on short passwords. A brute-force password cracker takes aim at a certain number of characters (usually no more than 4 or 5 characters) and then guesses every combination of typeable characters of the maximum length or less. This can be a long process. Oracle actually limits the number of typeable characters by converting all passwords to uppercase before hashing (this changes in 11g). In total, there are 68 different characters that can form an Oracle password. To do a brute-force search on four-character passwords, involves searching on all one-character passwords (68 of them), all two-character passwords (4624 of them), all three-character passwords (314,432 of them), and all four-character passwords (21,381,376 of them). That's a lot of passwords, and the number keeps going up exponentially as you add more characters to the password. At or beyond six characters, brute-force password cracking is generally ineffective, as it requires guessing billions of combinations.

It's best to assume that a password that meets any of the following criteria is weak:

- It appears in the English dictionary
- It is the name of a well-known city anywhere in the world
- It is the name of any professional sports team
- It is a calendar date
- It is a simple pattern, such as abcdef, 98765, or jjjjjj
- It is the same as the username
- It is less than six characters long

Who Can Remember a Strong Password?

Actually, it's worse than just remembering one password. You need a different strong password for every system. That's hard, particularly when you want to choose passwords like *wygc?gb!* or *gy7*ui9clor.* What you need is a system that allows you to generate seemingly random strings that actually aren't random at all. It all starts by picking a methodology or technology for choosing your passwords.

Password Management Tools

The best choice for managing a large number of strong passwords is to use a secure password management tool. Earlier we mentioned Password Safe from sourceforge.net (http://sourceforge.net/projects/passwordsafe). Other similar free password managers are Password Corral from Cygnus Productions (www.cygnusproductions.com/freeware/pc.asp) and KeePass from sourceforce.net (http://keepass.sourceforge.net/).

All three will generate strong passwords, and then store them for you in an encrypted file protected by a master pass phrase. With a tool like this, you can have different passwords for each system you use, but only need to remember the password to unlock the password manager. This is ideal, because the passwords for your databases can be randomly generated and impossible to remember. You only need to generate one strong password for yourself. If pass phrases are allowed (long strings with spaces), it is preferable to use a sentence. Pick your favorite line from a movie or part of the chorus of your favorite song, add some punctuation or mix in some uppercase letters and you've got yourself a passphrase. For example, this would make a nice pass phrase "It'S a SiciliaN message. IT means LucA brasi SleepS with ThE fishes."

Password Rule Sets

If you want to go it alone and remember all your passwords, there are ways to use strong passwords and remember them. If you've got a good memory, you can probably use a simple method, like choosing a phrase and picking the first letters of each word as a password. For example: "My daughter Marlee was born in August" becomes MdMwbi8, "I went to school at UMass Amherst" becomes "Iw2saUMA."

If memorizing many phrases and which systems they work with seems like a daunting task, there is another way. Gina Trapini wrote about Password Rule Sets on lifehacker.com. Rule sets make remembering lots of strong passwords fairly easy. The idea is to start with a base password, then customize the base password for each system.

Start with a complex looking but easy-to-type base password, something like iop[]\ (this is easy to type, the keys are all lined up in a row). Next add customization for each system. If you've got a SID named ORCL, create a password like ORiop[]\CL, LCiop[]\RO, or iop[]\CLOR. Even ORCLiop[]\ and iop[]\ORCL are pretty good passwords.

Make a rule for how you will customize the base password, then follow that rule for each system. If we picked ORiop[]\CL in the example above, a SID named HR would get the password Hiop[]\R and a SID named PEOPLESOFT would get the password PEOPLEiop[]\SOFT. In this case, the password rule splits the SID name and then places the base password in the middle. You really only need to remember the rule and the base password.

Passwords For Non-production Systems

It's important to take password management seriously on non-production systems as well as production systems. This goes beyond best practices and beyond a mindset for taking security seriously in all systems regardless of their value. The fact is, many non-production systems contain sensitive data that needs the same level of protection as is needed in a production environment.

The reality, however, is that most organizations completely ignore the security of their test and development databases. This is a big problem, as many of these ignored non-production systems contain a copy of production data or a copy of a production schema and configuration data. This is hacker gold. If you've got test systems that include even a partial copy of the data in your sensitive production systems, you need to secure those systems like production. That means removing default accounts and setting strong passwords to protect against unauthorized access. A hacker is just as happy to collect their booty from a test box as they are to collect it from a live production system. It's the data they are after.

Even if your non-production system only contains a copy of the production schema and perhaps database configuration, this needs to be protected. An attacker that can get this information can then use it as a roadmap to hack into the real production system where the sensitive data lives. It just makes sense, when it comes to protecting access to a database, to treat every one of them like production, and get the passwords and password controls locked down.

Password Controls Using Oracle Profiles

Oracle provides a robust set of security features around passwords and password controls. Options are available to lock accounts after a set number of failed logins, to expire passwords after a set period of time, and to enforce flexible complexity requirements on every new password added to the system. These features are all implemented within Oracle profiles.

The Oracle Profile

A profile allows an administrator to set limits on a user's resource utilization and to manage their password. You can create a number of profiles with different settings and then assign them to groups of users. A profile includes two groups of settings: *kernel limits* and *password limits*.

Kernel limits focus on controlling the resources a user can consume. There are settings for the maximum number of concurrent sessions a user can have, a maximum CPU time for a session and a call, a maximum idle time before a session is disconnected, a maximum read blocks per session and per call, and a whole slew of other resource-related items.

Password limits implement the security controls. Here are settings that control how complex a password must be, how long each password can be used before a change is required, and how many failed login attempts are allowed before an account is automatically locked. These settings enforce your password policies, and are the second line of defense against password crackers.

By default, there is one profile created, called DEFAULT. All users are assigned the DEFAULT profile unless otherwise specified. The DEFAULT profile is installed with all security options disabled, you must enable them on your own or create new profiles to enforce password controls.

Profile information is available in the DBA_PROFILES view.

```
SQL> describe dba_profiles;

Name                      Null?            Type
------------------------- ---------------- -------------------
PROFILE                   NOT NULL         VARCHAR2(30)
RESOURCE_NAME             NOT NULL         VARCHAR2(32)
RESOURCE_TYPE                              VARCHAR2(8)
LIMIT                                      VARCHAR2(40)
```

- PROFILE is the name of the profile being described, such as DEFAULT.
- RESOURCE_NAME is the parameter being described, such as FAILED_LOGIN_ATTEMPTS.
- RESOURCE_TYPE indicates if the parameter describes a KERNEL or a PASSWORD limit.
- LIMIT is the value of the parameter being described.

```
SQL> select PROFILE, RESOURCE_NAME, LIMIT from dba_profiles where RESOURCE_NAME =
'FAILED_LOGIN_ATTEMPTS';

PROFILE                 RESOURCE_NAME                 LIMIT
----------------------- ----------------------------- -----------------------
DEFAULT                 FAILED_LOGIN_ATTEMPTS         3
MONITORING_PROFILE      FAILED_LOGIN_ATTEMPTS         UNLIMITED
```

Failed Login Attempts

Of all the password controls in Oracle, this one is the most important. The FAILED_LOGIN_ATTEMPTS parameter in a profile indicates the number of failed log in attempts that are allowed before a user's status is automatically changed to LOCKED. This feature is commonly referred to as *account lockout*.

The feature is implemented as a simple counter. At first the count is set to 0. With each failed login attempt, the count is increased by 1 until the value for FAILED_LOGIN_ATTEMPTS has been reached, at which point the account is locked. Users cannot log in to locked accounts even with the correct password. An administrator must unlock the account or the system must be configured with a password lock time value that automatically unlocks accounts.

Account lockout features are an effective security measure against password crackers. By limiting the number of guesses the tool can make, you can dramatically reduce the risk posed by password attacks. If you configure your system for a reasonable number of failed login attempts, between two and five, it becomes almost impossible to mount a successful password-guessing attack. This is an important measure to take right away, especially if you have weak passwords in your system today, you can significantly reduce the risk by configuring FAILED_LOGIN_ATTEMPTS on every profile in your system.

Configure account lockout after three failed attempts in the DEFAULT profile with the following statement:

```
alter profile DEFAULT limit FAILED_LOGIN_ATTEMPTS 3;
```

Setting FAILED_LOGIN_ATTEMPTS to UNLIMITED disables the account lockout feature. There is no reason to disable account lockout in any Oracle database.

Password Life Time

The PASSWORD_LIFE_TIME parameter puts a limit on the number of days that a password can be used before it must be changed. This feature is often called *password expiration*, as it automatically forces users to change their password on a periodic basis. It works by prompting the user for a new password on the first login after a password has expired. Oracle will not allow the user to proceed without changing their password once their password lifetime (and password grace time) have expired.

Periodically changing passwords is a good security measure. This may keep an attacker guessing, or could take access away from someone who has stolen a password. It is not uncommon for an attacker to steal a large number of passwords and then crack them over time. If it takes 30 days to crack your password, but you change your password every 30 days, you can effectively limit an attacker's ability to gain unauthorized access to your account and system.

Configure password life time for 45 days in the DEFAULT profile with the following statement:

```
alter profile DEFAULT limit PASSWORD_LIFE_TIME 45;
```

Setting PASSWORD_LIFE_TIME to UNLIMITED disables the password expiration feature. While it may be necessary to have some accounts with passwords that do not expire, you should create a special profile for only those accounts and choose a strong password. Make sure account lockout is enabled and set to a low threshold.

Password Reuse Max

The PASSWORD_REUSE_MAX parameter controls the number of times a user's password must be changed before an old password can be reused. The intention is to limit reuse of passwords by forcing users to wait a long time before the system will accept the old password again. Delaying reuse of old passwords based on the number of password changes since an old password has been used, is confusing. Instead of using this setting, we recommend using Password Reuse Time instead. These two parameters work together. Setting to UNLIMITED voids the setting of the other and makes it so that passwords can never be reused. In order for either PASSWORD_REUSE_MAX or PASSWORD_REUSE_TIME to take effect, both must be set to an integer value.

In order to use PASSWORD_REUSE_TIME exclusively, set PASSWORD_REUSE_MAX to *0*.

```
alter profile DEFAULT limit PASSWORD_REUSE_MAX 0;
```

WARNING

In general, we do not recommend using the PASSWORD_REUSE_MAX parameter to control reuse of old passwords. When this feature is used in conjunction with the PASSWORD_REUSE_TIME parameter, things can get very confusing, as even after the reuse time has passed an old password still may not be able to be reused, because there have not been enough interim password changes to satisfy the reuse max parameter. Confusing enough?

When used alone (with PASSWORD_REUSE_TIME set to *0*), the PASSWORD_REUSE_MAX is generally useless. There is nothing stopping a user from changing their password repeatedly until the threshold is met and the system accepts an old password for reuse. Even if the value is set to 10 or 100, it takes only a few moments to change a password over and over again and completely defeat the feature.

Password Reuse Time

The PASSWORD_REUSE_TIME parameter controls the number of days a user must wait before reusing and old password. This ensures that password changes stick, and that a user cannot simply change their password to a new value, then immediately change it back to the old value. This setting works in conjunction with the PASSWORD_REUSE_MAX setting (see above for details).

We recommend that you configure the password reuse time to be one year or 365 days. Setting a value of UNLIMITED ensures that an old password can never be reused.

```
alter profile DEFAULT limit PASSWORD_REUSE_TIME 365;
```

Password Lock Time

The PASSWORD_LOCK_TIME parameter controls the number of days an account will remain locked as a result of the Failed Login Attempts limit being reached before the account is automatically unlocked. In general, there is no reason to automatically unlock accounts where password guessing may have been attempted. Instead, manually unlock accounts after verifying why the account became locked in the first place. Did the user forget their password, or was something more sinister going on?

In order to disable automatic unlocking of accounts, set the PASSWORD_LOCK_TIME to UNLIMITED.

```
alter profile DEFAULT limit PASSWORD_LOCK_TIME UNLIMITED;
```

Password Grace Time

The PASSWORD_GRACE_TIME parameter works together with PASSWORD_LIFE_TIME. The PASSWORD_GRACE_TIME represents the number of days after the password life time has expired before a user is forced to change their password, rather then being prompted with the option of changing their password. This setting is designed as a convenience for users, allowing the database to give them a gentle reminder that their password has expired and to please change it soon, before stepping in and demanding the password be changed immediately.

The PASSWORD_GRACE_TIME has limited security implications. For most systems, a setting of 5 to 15 days is reasonable; however, consider reducing the PASSWORD_LIFE_TIME if a password grace time is set. If no grace time is desired, set this parameter to *0*.

```
alter profile DEFAULT limit PASSWORD_GRACE_TIME 0;
```

Password Verify Function

The PASSWORD_VERIFY_FUNCTION parameter indicates which (if any) PL/SQL function will be used to check the complexity of a new or changed password. This is the most flexible of all the password controls, as it allows you to specify a function of your choosing to check password strength, before accepting the password into the database. By default, the PASSWORD_VERIFY_FUNCTION is disabled (set to NONE); however, Oracle comes with a sample verify function that performs a number of checks:

- Password must be a minimum of four characters in length
- Password must not equal the username
- Password must include at least one alphabet character, one numeric character, and one punctuation mark
- Password must not be welcome, account, database, user, password, oracle, computer, or abcd
- Password must differ from the previous password by at least three characters

There is nothing wrong with using the built-in Oracle function, although it clearly has room for improvement. The PASSWORD_VERIFY_FUNCTION must be present in the SYS schema. To install the default function, connect as sysdba and run the script in *$*ORACLE_HOME/rdbms/admin/UTLPWDMG.SQL. This will create the function *verify_function* which you can then specify to be used as follows:

```
alter profile DEFAULT limit PASSWORD_VERIFY_FUNCTION verify_function;
```

This function is written in PL/SQL and is well commented and easy to understand. It would be a simple task to tweak the function to your needs, increasing the minimum length to six characters and adding to the very basic password dictionary. Get a little fancier by reading the password dictionary out of a file or table, allowing for easy checking against a list of thousands of easily guessed passwords. You can find a sample implementation of a dictionary-based password verify function at http://orafaq.com/node/1027.

Assigning Profiles to Users

Once you have all your profile settings configured, you must assign each user to a profile. Unless otherwise specified, users are members of the DEFAULT profile. If you made your configuration changes to the DEFAULT profile, you do not need to assign the profile to your users. However, if you created a new profile, or multiple profiles based on user roles, assign the profile to users with the following SQL statement:

```
alter user SCOTT profile DEFAULT;
```

If you have created a new profile but have not specified values for all parameters, those unspecified will revert to the DEFAULT value, which is the value programmed for the same parameter in the DEFAULT profile. When you list profile settings by selecting from *dba_profiles*, it is not unusual to see parameter values set to DEFAULT.

```
SQL> select PROFILE, RESOURCE_NAME, LIMIT from dba_profiles where RESOURCE_TYPE =
'PASSWORD' and PROFILE = 'MONITORING_PROFILE';

PROFILE                RESOURCE_NAME                LIMIT
---------------------- ---------------------------- -----------------------
MONITORING_PROFILE     FAILED_LOGIN_ATTEMPTS        UNLIMITED
MONITORING_PROFILE     PASSWORD_LIFE_TIME           DEFAULT
MONITORING_PROFILE     PASSWORD_REUSE_TIME          DEFAULT
MONITORING_PROFILE     PASSWORD_REUSE_MAX           DEFAULT
MONITORING_PROFILE     PASSWORD_VERIFY_FUNCTION     DEFAULT
```

```
MONITORING_PROFILE      PASSWORD_LOCK_TIME          DEFAULT
MONITORING_PROFILE      PASSWORD_GRACE_TIME         DEFAULT
```

You can determine a users profile by selecting from the *dba_users* view:

```
SQL> select USERNAME, PROFILE from dba_users where USERNAME = 'SCOTT';

USERNAME                PROFILE
----------------------- -------------------------
SCOTT                   DEFAULT
```

Are You 0wned?

Remote OS Authentication

Oracle offers a "feature" to allow clients to establish remote connections to the database while relying entirely on the client's operating system (OS) to authenticate the database user. This feature is controlled by an initialization parameter called *remote_os_authent*; setting the parameter to TRUE enables remote OS-authenticated connections. If you've got this feature enabled on any of your databases, you need to ask yourself, am I owned?

With remote OS authentication enabled, it would be trivial for an attacker to compromise any users in your database that are able to authenticate via the OS. The worse case scenario here would be if the OS authentication prefix is set to null. This would allow an attacker to load an Oracle client on any machine on the network, and start that client with a local OS user they create, such as SYSTEM. Point the client at the database in question and log in using *sqlplus /*.

Oracle will recognize the client username as *SYSTEM*, it will add the OS prefix, in this case null, and then it will check for the resulting account name in the database user list. It won't run this exact query, but it's the concept that matters:

```
select NAME from SYS.USER$ where NAME = 'SYSTEM';
```

Of course a match will be found, and the user will be authenticated to the database as SYSTEM. Perhaps this is an extreme example. Let's consider a system that has *os_authent_prefix = OPS$*. If an attacker can gain any access to the target database, he or she can get a list of users by selecting from the ALL_USERS view.

Continued

```
C:\>sqlplus scott/tiger

SQL*Plus: Release 10.2.0.1.0 - Production on Sat Sep 15 13:18:52 2007

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> select username from all_users;

USERNAME
------------------------------
OPS$SEAN
OPS$ERIC
BOOTSY
SRS
ALLAN
AARON
MARLEE
JILL
JOSH
HAVIV
JONATHAN
```

You can see that two users are listed that are set up for OS authentication (*OPS$SEAN* and *OPS$ERIC*). If either of these OS users have more privilege than SCOTT, the attacker has not elevated their level of access. Remember, with remote OS authentication enabled, the attacker only needs to set up an account with the right name in his system (in this case SEAN or ERIC) and then connect to the database without specifying a username or password.

If it seems too easy, that's because it is! Make sure you have *remote_os_authent* set to FALSE on all of your databases.

OS Authentication

Oracle allows for other methods of authentication that reside outside the database. The most commonly used is Local OS Authentication, facilitating database connections for users of the database host OS. OS authentication has been part of Oracle since day one. In fact, that was the only authentication mechanism supported in the earliest releases of the database. OS authenticated access to the database is a good thing, particularly since most of the operating systems used to host Oracle databases have strong authentication and password management mechanisms.

OS authentication can also pose some serious risks to your system, particularly if things are not configured properly. It's actually all fairly simple, but some parts are definitely not intuitive. With an understanding of how to set up and identify OS authenticated accounts, and how to enable the security features around OS authentication, you can both enable and secure local OS authenticated access to your databases.

Remote OS Authentication

Remote OS Authentication is a feature that allows Oracle databases to blindly accept connections from users authenticated by a remote OS (meaning logged into a system that is not the database server). Remote OS Authentication represents a significant security risk to any database and should be disabled on all systems, both production and non-production.

This is an issue of trust. How can you trust any remote system to properly authenticate users. You have no way to verify that the remote system is configured securely, nor can you tell if it has been taken over by a hacker. If users need to connect with a remote client, allow them to authenticate to the database with a username and password. OS authentication is useful for local connections only.

Disable Remote OS Authentication by setting the REMOTE_OS_AUTHENT initialization parameter in *init.ora* to *FALSE* and restarting the database. Note that if you currently have remote clients connecting to the database with remote OS authentication, those connections will no longer work.

OS Authentication Prefix

Oracle provides a means for tagging OS authenticated users by adding a prefix to their OS username in order to form their database username. This prefix is called the OS Authentication Prefix, and is configurable to any value you like. The value is

controlled by the initialization parameter *os_authent_prefix* in *init.ora*, and is set to *OPS$* by default. If you decide to change this value, make the change when you first configure the database and then stick with it. Changing the OS Authentication Prefix on a working system can take a lot of effort.

When a user wants to make an OS-authenticated connection to an Oracle database, they use the following syntax, with no username or password specified:

```
sqlplus /
```

The database automatically captures the OS username, prepends the OS authentication prefix, and checks the value against the list of users in the database. If there is a match, authentication succeeds. No password check is required; the OS already checked the password. If there is no match, Oracle responds with the standard failed login message:

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

There is nothing wrong with using the *os_authent_prefix* default setting of *OPS$*. There would, however, be a problem with changing the value to null:

```
os_authent_prefix = ""
```

A null OS authentication prefix removes the tag on OS-authenticated users. This can lead to a real security issue where OS users can be created with the same name as an existing (and powerful) database user. Those users can then connect to the database without a password. The situation is significantly worse if remote OS authentication is enabled. If an attacker could create an OS user named SYSTEM or OUTLN or any number of other usernames, and the OS Authentication Prefix was set to null, they could connect to the database without a password, no matter how strong of a password was set for SYSTEM, OUTLN, or whatever other user is attacked.

Be sure that you have set the OS Authentication Prefix to some value. Even if you never changed the default setting, check it out and be sure, as the risk of a null prefix is too great to ignore.

```
SQL> select NAME, VALUE from v$paramter where NAME = 'os_authent_prefix';

NAME                                              VALUE
------------------------------------------------- --------------------------
os_authent_prefix                                 OPS$
```

WARNING

Beside the OS Authentication Prefix, Oracle does not differentiate very well between OS-authenticated users and regular password-authenticated users. In fact, you can set a password for an account that was originally set up as OS authenticated, and then connect to that account using either OS authentication or Oracle authentication with the password.

This is what makes the attack possible when the OS authentication prefix is null. Users that were created as Oracle authenticated users, can also be authenticated by the OS. Oracle simply doesn't automatically enforce the method by which a user was intended to authenticate to the database.

Creating and Identifying OS Authenticated Users

It's simple to create OS authenticated users. In general, it's also simple to identify the OS authenticated users you currently have in your databases.

To create an OS-authenticated Oracle database user account for an OS user named *PNUT*, use the following command (assuming *os_authent_prefix = OPS$*):

```
create user OPS$PNUT identified externally;
```

The key here is the *identified externally* part. That tells Oracle to mark the account as an OS-authenticated account and to only accept OS-authenticated connections for that account. Note that the addition of the prefix *OPS$* is not automatic. You must know the prefix and manually add it when you create the user, otherwise the OS-authenticated connection will fail.

Check out the entry for this user in *dba_users*:

```
SQL> select USERNAME, PASSSWORD from dba_users where USERNAME = 'OPS$PNUT';

USERNAME                                 PASSWORD
---------------------------------------- ------------------------------------
OPS$PNUT                                 EXTERNAL
```

There are two key elements here that identify this user as OS-authenticated. First and foremost is the username that starts with *OPS$*, next is the password that is set to *EXTERNAL*. This is the way you want to see the password set for all of your OS-authenticated users, as this ensures they can only log in to the database with an OS-authenticated connection, not with the option of OS or Oracle authentication. It is

definitely a good idea to find all the users in your database that are set up for OS authentication and make sure their password is *EXTERNAL*. The following query will give you a list of users set up for OS authentication, but who also have an Oracle password (assuming *os_authent_prefix = OPS$*):

```
select USERNAME from dba_users where USERNAME like 'OPS$%' and PASSWORD <>
'EXTERNAL';
```

Automated Scanning for Weak Passwords

You can take big steps by teaching folks how to create strong passwords, and by implementing the password control settings in Oracle profiles. What those measures do not do, however, is help you to identify and eliminate the weak passwords that are already in use in your systems. In order to do this, you will need an automated tool. Manually driven weak password detection is really not feasible. Luckily, there are good tools out there to help you that are available as both free open source versions, and fully supported commercial products.

Tools & Traps…

Oracle Password Scanners

Oracle password scanners or crackers come in various packages. One mechanism for characterizing these tools is to examine how they operate, and what information is required to get them going. Some scanners work by repeatedly attempting to log on to an Oracle system. They may cycle through a dictionary of passwords, or brute force scan based on a number of characters, or check for default accounts or accounts with username = password. Whatever mechanism they use to pick a username and password is not important for this characterization, it's that they attempt to log on to the database to determine if a username/password combination is valid.

This method of password scanning is powerful as it requires almost no knowledge. Point at an Oracle database and off it goes. If you have a list of usernames, it can do even better, targeting only those accounts that exist. This kind of zero knowledge scanning is particularly useful to an attacker coming at the database from the outside. The downside of this kind of outside-in scanning is that it puts a large load on the database server to process all those failed login attempts.

Continued

The other noticeable problem with this approach comes into play when account lockout is enabled. It would be easy for a password scanner to lock out every single account in an Oracle database with only a few seconds worth of password guessing. If you are going to run a password scanner that works by repeated login attempts, make sure you temporarily disable account lockout or face the consequences of locking lots of accounts. One nice thing to know is that it's not possible to lock a *sysdba* account, so you can always connect to a system as *sysdba* and unlock all the users you locked out.

A second mechanism for password scanning requires more knowledge; either a single username/password hash pair, or access to a list of usernames and password hashes (usually accessed vie the DBA_USERS view). These tools work offline, making their password guesses, but instead of submitting login requests to the database, they calculate the password hashes on their own and compare to the known value.

These inside-in password scanners are the most powerful tools for finding weak passwords and are also the most efficient. Since only valid users are scanned, the time of the scan is reduced. More importantly, this analysis phase of this kind of scan can be performed on a dedicated scanning machine instead of on the database server. This means almost no performance impact on the database for guessing passwords all day and night.

If you are generally going to have access to log in to the databases you want to scan for weak passwords, you should find a tool that scans inside-in, collecting the list of usernames and password hashes from the database and using them to verify the existence of weak passwords, by scanning offline. If you have no other choice, outside-in password scans will often reveal a default account that can then be used for a more efficient inside-in scan.

Freeware Password Scanners

The freeware scanning tools out there are great for very technical users that deal with small environments of databases. Commercial products are more flexible and scalable and tend to be much simpler to use. If you use freeware scanners, keep in mind that you are basically using a hacker tool with no guarantees of what could result. Be careful with your selection of freeware password scanners. Make sure to select one that is widely used and well known.

Probably the best and fastest free password scanner for Oracle is checkpwd from red-database-security. It comes packaged in a few varieties, some including all of the necessary client drivers and a very large password dictionary. Download checkpwd at http://www.red-database-security.com/software/checkpwd.html. Checkpwd is a command-line tool and output from running the tool looks like this:

```
C:\>checkpwd system/strongpw@//123.34.54.123:1521/ORCL password_list.txt

Checkpwd 1.23 [Win] - (c) 2007 by Red-Database-Security GmbH
Oracle Security Consulting, Security Audits & Security Training
http://www.red-database-security.com

initializing Oracle client library
connecting to the database
retrieving users and password hash values
opening weak password list file
reading weak passwords list
checking passwords
Starting 2 threads
MDSYS has weak password MDSYS [EXPIRED & LOCKED]
ORDSYS has weak password ORDSYS [EXPIRED & LOCKED]
DUMMY123 has weak password DUMMY123 [OPEN]
DBSNMP OK [OPEN]
SCOTT has weak password TIGER [OPEN]
CTXSYS has weak password CHANGE_ON_INSTALL [EXPIRED & LOCKED]
SH has weak password CHANGE_ON_INSTALL [EXPIRED & LOCKED]
OUTLN has weak password OUTLN [EXPIRED & LOCKED]
DIP has weak password DIP [EXPIRED & LOCKED]
DUMMY321 has weak password 123YMMUD [OPEN]
[...]
SYS OK [OPEN]
SYSTEM OK [OPEN]

Done. Summary:
Passwords checked : 13900828
Weak passwords found : 23
Elapsed time (min:sec) : 0:54
Passwords / second : 265486
```

Another free password scanning tool is orabf from toolcrypt.org. orabf can run both a dictionary-based scan from a file, as well as a brute-force attack with a specified number of characters. In order to run orabf, you need a username and it's password hash. orabf will try to crack it. orabf is a command-line tool, and is easy to run. This is what it looks like:

```
C:\orabf>orabf 79805A2064887852:DBSNMP -n 4

orabf v0.7.6, (C)2005 orm@toolcrypt.org
---------------------------------------
Trying default passwords...done

Starting brute force session using charset:
#$0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_
```

```
press 'q' to quit. any other key to see status

current password: BAJIE
4600301 passwords tried. elapsed time 00:00:06. t/s:721616

current password: CZNFM
8402692 passwords tried. elapsed time 00:00:11. t/s:722812

current password: EZCWF
13013499 passwords tried. elapsed time 00:00:18. t/s:722972

current password: IQY#F
21887793 passwords tried. elapsed time 00:00:30. t/s:707484

current password: KY60E
26824451 passwords tried. elapsed time 00:00:38. t/s:704747

177301218 passwords tried. elapsed time 00:04:22 t/s:674630
```

There are other free Oracle password scanners out there including John the Ripper (http://www.openwall.com/john) and Cain and Able (http://www.oxid.it/cain.html). Remember, these are powerful tools and you run them with no warranty or support. Use them on test systems before running in a production environment.

Commercial Password Scanners

You're probably not going to find all that many commercial products that do nothing but password scanning. Generally password scans are integrated into complete vulnerability scanning systems. There are a small number of commercially available vulnerability scanners that are focused on in-depth assessment of databases. These highly specialized tools can find almost any vulnerability or misconfigurations in your databases. They all offer powerful password scanning capability, some with both brute-force and dictionary scan features. Figure 7.2 shows Application Security, Inc's AppDetective Pro.

Figure 7.2 Application Security, Inc.'s AppDetectivePro Password Scan Results

High	Easily-guessed database password	Oracle10g Database (testing) (User Name=JOSH) (Password=patriots)
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=MARLEE) (Password=baby)
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=BOOTSY) (Password=funk)
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=OUTLN) (Password=outln) (Status=Lock
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=MDSYS) (Password=mdsys) (Status=Loc
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=ORDSYS) (Password=ordsys) (Status=Lc
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=CTXSYS) (Password=change_on_install)
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=XDB) (Password=change_on_install) (Sta
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=ORDPLUGINS) (Password=ordplugins) (S
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=OLAPSYS) (Password=manager) (Status=
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=BI) (Password=change_on_install) (Statu
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=AARON) (Password=mets)
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=SCOTT) (Password=tiger) (Status=Locke
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=HR) (Password=change_on_install) (Stat
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=PM) (Password=change_on_install) (Stat
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=SH) (Password=change_on_install) (Stat
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=DIP) (Password=dip) (Status=Locked)
High	Easily-guessed database password	Oracle10g Database (testing) (User Name=OE) (Password=change_on_install) (Stat
High	Password for database user same as username	Oracle10g Database (testing) (User Name=BBAKALA) (Password=BBAKALA)
High	Password for database user same as username	Oracle10g Database (testing) (User Name=PIC) (Password=PIC)
High	Password for database user same as username	Oracle10g Database (testing) (User Name=OUTLN) (Password=OUTLN) (Status=Lo
High	Password for database user same as username	Oracle10g Database (testing) (User Name=MDSYS) (Password=MDSYS) (Status=Lc
High	Password for database user same as username	Oracle10g Database (testing) (User Name=ORDSYS) (Password=ORDSYS) (Status=
High	Password for database user same as username	Oracle10g Database (testing) (User Name=EXFSYS) (Password=EXFSYS) (Status=L
High	Password for database user same as username	Oracle10g Database (testing) (User Name=WMSYS) (Password=WMSYS) (Status=L
High	Password for database user same as username	Oracle10g Database (testing) (User Name=ORDPLUGINS) (Password=ORDPLUGIN
High	Password for database user same as username	Oracle10g Database (testing) (User Name=SI_INFORMTN_SCHEMA) (Password=S
High	Password for database user same as username	Oracle10g Database (testing) (User Name=TSMSYS) (Password=TSMSYS) (Status=
High	Password for database user same as username	Oracle10g Database (testing) (User Name=MDDATA) (Password=MDDATA) (Status

The top commercial products that focus on database vulnerability scanning are Application Security, Inc.'s DbProtect and AppDetectivePro (www.appsecinc.com) and Next Generation Security Software's NGSSQuirreL (www.ngssoftware.com). Both products have great password scanning capabilities, allowing you to run dictionary scans, brute-force scans, check for username = password, and do it all quickly and efficiently. Both products support other commercial database platforms as well, giving you options for scanning the non-Oracle databases within your organization.

Summary

Password crackers represent a significant risk to Information Technology (IT) systems that protect themselves with no more than a username and password. Powerful password scanners are freely available for download in various places on the Internet, allowing even the least experienced hackers the ability to break into what should be secured systems.

Oracle provides a number of security features to protect against password attacks. The features are easy to implement and can turn a database from an easy cracking target into a hardened fortress. Strong password policies are a key part of any security plan. Establish a system for managing and generating passwords, and teach your users how to take advantage of it.

The education process is important here. You can't just enable a complex password verify function, expire everyone's password, and expect things to work out. Users may be unable to pick an acceptable password or may choose something they instantly forget. In either case it means more work and hassle for everyone. Make a plan for better passwords, distribute it, and then take the step to have users reset their password based on the new standards. Security is a process, and everyone has to learn to be part of it.

In order to protect your systems, you should periodically run a password-cracking scan and change any weak passwords that are discovered. Both freeware and commercial tools are available, the choice is yours. A system with weak passwords and minimal password controls is like a safe with the combination taped to the lock. The safe looks imposing and secure from a distance, but once you get close it's a piece of cake to get inside. Make the lock on the door to your Oracle database work for you—lock down your passwords and password controls.

Solutions Fast Track

Configuring Strong Passwords

- ☑ Passwords protect access to the database. Because of this, they are a prime target for attackers. There are literally hundreds of freely available password cracking tools out there, many of which can be used to attack Oracle databases.

- ☑ Weak passwords are those that are easy to guess, either for a human or a computer. Passwords are weak if they are dictionary words, usernames, sports teams, dates, or patterns.
- ☑ Use a password management tool to generate and store strong passwords for all your systems. If you want to try to generate your own strong passwords and remember them, set up a password rule set.

Password Controls Using Oracle Profiles

- ☑ Enable Oracle's account lockout feature by configuring the FAILED_LOGIN_ATTEMPTS parameter in every profile in the database. This feature stops password guessing attacks by locking the account after a set number of failed logon attempts.
- ☑ Use the PASSWORD_LIFE_TIME profile feature to automatically expire users passwords, forcing them to choose a new password. The new password must meet all the criteria enforced by Oracle's password controls.
- ☑ Use the PASSWORD_VERIFY_FUNCTION to check the strength of new passwords. Oracle provides a default function that can be used or easily modified to provide additional protections. There are even better sample functions available on the internet.

OS Authentication

- ☑ Oracle has native support for accepting connections to the database that are authenticated by the OS rather than the database. The OS-authenticated connections are a good thing, but must be configured properly to avoid security problems.
- ☑ Be sure to disable remote OS authentication on all your databases. When set to TRUE, this configuration allows Oracle to accept connections that are authenticated by a remote OS rather than the database server. Since it is impossible to trust any remote OS, this represents a huge risk.
- ☑ Configure an OS authentication prefix to help identify accounts that are set up for OS authentication, and to block simple attacks where an attacker uses OS authentication to connect to Oracle using an account that was meant to be accessed via Oracle authentication only.

Automated Scanning for Weak Passwords

- ☑ Manually searching for weak passwords is an impossible task, even in a single database. There is simply no mechanism to decipher the password hash values stored in the database to determine the original password.
- ☑ Automated tools that scan Oracle databases for weak passwords are widely available. There are both freeware scanners and commercial products available that target databases. The free software is better suited for small environments run by tech-savvy users. The commercial products are better for scanning large numbers of systems with an easy-to-use interface.
- ☑ Oracle password controls only impact new passwords, they do not go back and flag existing passwords that don't meet the new standards. It's important to scan all systems for weak passwords on a regular basis, and to move aggressively to have them changed.

Frequently Asked Questions

The following Frequently Asked Questions, answered by the authors of this book, are designed to both measure your understanding of the concepts presented in this chapter and to assist you with real-life implementation of these concepts. To have your questions about this chapter answered by the author, browse to **www.syngress.com/solutions** and click on the **"Ask the Author"** form.

Q: Can you explain how to configure Oracle's account lockout feature?

A: Set the FAILED_LOGIN_ATTEWMPTS parameter in a profile (or all profiles) to a numeric value (the value UNLIMITED disables account lockout). Associate any or all users with this profile. You can modify this value in the built-in DEFAULT profile, which will cover all users that have not been explicitly associated with a different profile. Here is an example:

```
alter profile DEFAULT limit FAILED_LOGIN_ATTEMPTS 3;
alter user SCOTT profile DEFAULT;
```

Q: Do I really need to configure strong passwords on my non-production systems? It's much easier to set easy to remember passwords.

A: Yes! Many non-production systems hold sensitive data. This may be a copy of production data being used in development or for testing, or it may be information about the production schema and configuration. An attacker is just as happy to steal your secrets from your development lab as she is to steal them from your fully locked down production data fortress. Furthermore, if an attacker can take over the non-production server via the database (lots of attacks allow this to happen), they can use it as an internal point to gain reconnaissance and launch attacks against other internal systems. It's important to configure strong passwords on all your Oracle systems.

Q: I've seen a few Oracle attack demos where the attacker is trying to get a copy of the usernames and passwords from the DBA_USERS view. Aren't these passwords secured with some kind of encryption?

A: The passwords stored in DBA_USERS (actually stored in SYS.USER$) are stored securely. They are not exactly encrypted; they are hashed, which is a cryptographic operation similar to encryption except that a hash is a one-way function. There is no way to take an Oracle password hash from DBA_USERS and work backwards to the user's password. There is, however, a simple means to take a whole lot of passwords and work forwards to create an Oracle password hash. Compare the hash you calculate with the value stored in the database. If you find a match, you know the user's password. There are many freely available tools out there that automatically perform this kind of password cracking on Oracle password hashes. Make sure to keep any non-DBA users out of Oracle's list of database passwords.

Q: Do you have any tricks to use for remembering strong passwords?

A: The best trick is to use a password management tool; there are a few good ones out there that you can get for free. If you want to remember them yourself, find something you can easily remember and use it to build a strong password. For example, take a phrase such as "I was born in Boston, MA in February of '76," and use the first letter of each word, plus all the numbers and punctuation to make a strong password: IwbiB,MiFo'76.

Q: What is a Password Verify Function and how do I use it?

A: The Password Verify Function is Oracle's means of providing controls on password complexity. It allows an administrator to specify a PL/SQL function that will be used to check the complexity of every new password set in the database. The function must conform to a defined application program interface (API), and must exist in the SYS schema. Otherwise, it can do whatever you please. Oracle packages a sample function in the file *utlpwdmg.sql*, (found in *$ORACLE_HOME/rdbms/admin*). Run this script to install the default *verify_function*. This provides some basic complexity checks, which you can easily improve or tailor to your organizations needs.

Q: What is the OS_AUTHENT_PREFIX and why is it important?

A: OS_AUTHENT_PREFIX is a configuration parameter that allows you to specify a value that will be automatically prepended to the usernames that authenticate via the OS. By default, OS_AUTHENT_PREFIX is set to *OPS$*. Therefore, for an account to connect to the database with OS authentication, that account name with the proper prefix must exist as a user in the database. Setting OS_AUTHENT_PREFIX to null creates a security hole where no prefix is added to OS usernames logging into the database. Since Oracle trusts the OS to check the users password, all it does is verify that the proper username exists in the database. If you create an OS user named SYSTEM and the OS_AUTHENT_PREFIX was null, that OS user could log in as the database user SYSTEM without any password checking done by Oracle. It's really important to set OS_AUTHENT_PREFIX to a non-null value.

Q: How often should I scan my production systems for weak passwords? What about my non-production systems?

A: It's a good practice to scan all of your production databases for weak passwords on a monthly basis. Non-production systems should also be scanned regularly, ideally just as often as production. Having a commercial database vulnerability scanner that can automatically run large scan jobs across many databases makes it easy to scan all systems with nearly any frequency that you desire. At a minimum, be sure to scan your non-production systems for weak passwords on a quarterly basis.

Q: How often should I check the password control settings on my databases?

A: Ideally, you would only need to check them once. Get them set correctly, and only re-check them when and if your database monitoring system detects a change to one of the settings. This actually is possible, but you'll need to purchase a complete database security platform to get it. More realistically, check the password controls on all of your systems at least once a quarter; it's even better to check once a month.