

S E C U R I T Y



Ron Ben Natan

Implementing Database Security and Auditing

Includes
Examples For:

**ORACLE
SQL SERVER
DB2 UDB
SYBASE**

Authentication and Password Security

In Chapter 1, you learned about secure installations of your database and that you should fully understand and use the built-in mechanisms within your database—mechanisms that help you authorize and enforce activities within your database. However, in order to authorize and enforce, you must be able to first identify the party that is requesting the action. This identification process is closely linked to the authentication process—the process in which the server can prove to itself that the requesting party is who it claims to be. Authentication and various related topics are the subject of this chapter.

Authentication forms the basis of any security model, as shown in Figure 4.1. If you cannot authenticate a user, how can you assign any privileges? The SANS glossary (www.sans.org/resources/glossary.php) defines authentication as “the process of confirming the correctness of the claimed identity”—it is the process where an entity provides proof that it is who it is claiming to be. The issue of identity is separate from authentication, and several methods are used to define an identity. Methods by which you can identify a party include the following:

- Something that the party knows (e.g., username and password)
- Something that the party possesses (e.g., a badge, smart card, or certificate)
- Some biometric attribute that the party has (e.g., fingerprints or a retinal pattern)

The focus of this chapter is on the authentication process, and I will always use the username/password identity-creating method. Usernames and passwords are by far the most common methods you will encounter.

→
Figure 4.1
*Authentication as
the base of the
security model.*



Also, from your perspective, there really is no difference what identity method your organization is using, and the differences will be transparent to the database environment, because they will all be taken care of in lower levels of the software stack. The identity is merely something that the party signing on has, and the authentication process is that in which you inspect what the entity has and decide if this proves that they are who they say they are.

The first part of this chapter introduces you to the various authentication categories that the main database vendors support. You will learn what authentication options make your environment inherently insecure and what type of authentication options you should consider. You should always remember that if your authentication setup is insecure, nothing else matters. Once you understand how to configure for strong authentication, you will also learn what activities you should perform on an ongoing basis to ensure that authentication and identities remain secure.

4.1 Choose an appropriate authentication option

Every database has an authentication procedure—the procedure by which a user is challenged to provide a set of credentials and in which the database verifies whether the user is who they claim to be. Once authenticated, the database knows who the user is and can assign a set of privileges, but this is already outside the scope of authentication and is part of the authorization mechanism.

4.1.1 Anatomy of the vulnerability: Weak authentication options

Most databases will allow you to control how authentication is done. This means that if you're not careful and don't understand all the implications, you could end up with rather weak authentication (i.e., a gaping hole in the security of your database).

Let's look at an example from the world of DB2 UDB. DB2 allows you to choose among several authentication options. One of the options is called CLIENT, but it may as well have been called "no authentication." CLIENT authentication in DB2 UDB means that the database performs no authentication on the server. If it gets a connection request, it assumes that authentication has already happened on the client and accepts the credentials from the client without doing any further authentication. This is a bad assumption because it allows me to plug into the network and almost instantaneously connect to the DB2 instance without anyone having really checked me out.

CLIENT authentication in DB2 assumes that people protect the client workstations—a bad assumption. It has a concept of TRUSTED CLIENTS representing all clients that have a "true" operating system, which can perform authentication. For example, a Windows 9x machine will never be a trusted client. However, the issue is not so much whether the OS can authenticate or not (it may have been so seven years ago, but no more); the issue is that workstations and laptops are not always a good security environment, and it is dangerous to rely on authentication at the endpoints for your database security. Just out of interest, I recently went around the group I am working with at a client to see what their passwords were like. I asked five people in sales and eleven people in technical support. Four of the five people in sales had a Windows account with no password whatsoever. The support people were a little better in their own accounts, but all the machines they were working on had a privileged account with the same password that was well known and easy to guess. The support people used this because they all had a need to sign on to each other's machines to run tests or troubleshoot issues. The system administrator passwords on these machines were good, but two of the support people had the password written on a sticky note stuck to the monitor because it was so difficult to remember. Do you really want to trust your database to that kind of an authentication environment?

4.1.2 Implementation options: Understand what authentication types are available and choose strong authentication

Most databases have more than one authentication option that you can set up and use. Some databases have a very large set from which you can choose. Choice is generally a good thing, but it does put the burden on you to choose wisely. What you should take away from the example in the previous subsection is that it is very important that you know what authentication options are available within your database environment and use one that truly authenticates users trying to access the database.

Let's continue with the DB2 UDB example started in the previous subsection and see what a better authentication option might look like. But first a quick word on the DB2 UDB authentication layer. DB2 UDB does not have its own authentication implementation (as do Oracle, SQL Server, and Sybase). DB2 UDB *must* rely on an external authentication system, most commonly the operating system. For example, when you install DB2 UDB on a Windows system, it automatically creates a new Windows user for the database administrator, as shown in Figure 4.2. At first this may seem limiting to you, especially if you're used to another database environment. As it turns out, most vendors (including Oracle and Microsoft) actually recommend operating system–based authentication because it is usually a stronger authentication model and usually provides better overall security.

DB2 UDB CLIENT authentication should never be considered plausible—at least not with its related defaults. Two additional attributes can help you refine CLIENT authentication. The first, `TRUST_ALLCLNTS`, can be set

Figure 4.2
A Windows user is created when installing DB2 in Windows, because DB2 UDB uses the operating system to authenticate users.



to a value of `DRDAONLY`, which means that the server will authenticate all clients except those coming from z/OS, OS/390, VM, VSE, and iSeries operating systems—environments considered to be far more secure and controlled than clients on Windows or UNIX. The second parameter is called `TRUST_CLNTAUTH`, and it determines where a password should be checked when clients are authenticated. The parameter can be set to `SERVER` or `CLIENT`, and the value determines if the passwords are checked on the client (where the DB2 driver runs) or the server. If you have decided to go with `CLIENT` authentication, I strongly suggest you set `TRUST_ALLCLNTS` to `DRDAONLY` and `TRUST_CLNTAUTH` to `SERVER`. Unfortunately, `TRUST_ALLCLNTS` is set to `YES` by default, meaning that if you do set the authentication mode to `CLIENT`, your DB2 instance will trust all connections.

`CLIENT` is not the default authentication option for DB2 UDB, so you have to explicitly change it to this weak mode. I know I'm being repetitive, but please don't use `CLIENT` authentication.

The default authentication mode used by DB2 UDB is called `SERVER` authentication. This option specifies that authentication occurs on the database server and uses the server's operating system security layer. Note that because the database server's operating system is used to authenticate the user, any local connection (i.e., one initiated from the database server) does not go through any authentication phase at the database level—there just is no point. `SERVER` is not only the default authentication option, it is also by far the most common. Other authentication options supported by DB2 UDB 8.2 are as follows:

- `SERVER_ENCRYPT`. Authentication happens at the server but requires the client to pass encrypted usernames and passwords.
- `KERBEROS`. Used when the operating systems of both the client and the server support the Kerberos security protocol. Kerberos is an important authentication system and one that has gained widespread usage in the industry for a variety of systems. (See Appendix 4.A for an overview of Kerberos.)
- `KRB_SERVER_ENCRYPT`. Used to allow the server to accept either Kerberos authentication or encrypted server authentication.
- `DATA_ENCRYPT`. Authentication is exactly like `SERVER_ENCRYPT`, but the entire session is encrypted over the wire. Note that this feature is new to UDB 8.2 and was not available previously.

- `DATA_ENCRYPT_CMP`. Authentication is like `SERVER_ENCRYPT`, and communication will use `DATA_ENCRYPT` if the client supports it with a fallback to unencrypted communications if the client does not.
- `GSSPLUGIN`. This is also a new feature in UDB 8.2 allowing an extensible authentication approach. You can plug in any authentication mechanism that conforms to the GSS API to become UDB's authentication provider.
- `GSS_SERVER_ENCRYPT`. Authentication is either `GSSPLUGIN` or `SERVER_ENCRYPT`.

You've now seen that DB2 UDB uses the server OS for authentication, and I mentioned that this is often also the recommended authentication option in other database environments. The main reason that operating system authentication is a good option is that it solves the credentials management issue; it allows you to let the operating system take care of credential management rather than having to carefully consider where and how you store user credentials. Let's move on to look at the authentication options for SQL Server and Oracle.

Microsoft SQL Server has two authentication modes: Windows authentication and mixed authentication. Windows authentication is the default mode and the one recommended by Microsoft. Windows authentication means that SQL Server relies exclusively on Windows to authenticate users and associate users with groups. Mixed authentication means that users can be authenticated either by Windows or directly by SQL Server. In this case SQL Server still uses Windows to authenticate client connections that are capable of using NTLM (NT LAN Manager) or Kerberos, but if the client cannot authenticate, then SQL Server will authenticate it using a username and password stored directly within SQL Server. NTLM is an authentication protocol used in various Microsoft network protocol implementations and is used throughout Microsoft's systems as an integrated single sign-on mechanism.

Let's move on to Oracle. Oracle also has many authentication options, including native Oracle authentication, which uses Oracle tables to maintain passwords, and operating system authentication. Let's start by understanding how native authentication works using a simple example showing an interaction between a client using OCI and an Oracle server.

The native authentication process starts when a client asks you for a username and password and calls the OCI layer. At this point the Transparent Network Substrate layer (TNS) is called. TNS makes a network call to

the server and passes it some client identifiers, like the hostname and an operating system name. It does not pass the username and password yet; rather, it calls a system call at the operating system level and retrieves the operating system user that is being used. The database does not try to authenticate this operating system username; it just accepts this information and proceeds to negotiate an authentication protocol with the database (all within the TNS layer). When the two agree to an authentication method, the client sends the login name and password to the database using the Oracle Password Protocol (also called O3LOGON)—a protocol that uses DES encryption to ensure that the password cannot be easily retrieved by an eavesdropper.

Note that this means that for every connection, the database knows the user not only at the database level but also at the operating system level. This information may be important to you for audit or security purposes, and you can retrieve it from `V$SESSION`. For example, the following data fields are taken from `V$SESSION` and can be useful when you want to better categorize who is logged into the database:

```

USERNAME:      SYSTEM
OSUSER:        RON-SNYHR85G9DJ\ronb
MACHINE:       WORKGROUP\RON-SNYHR85G9DJ
MODULE:        SQL*Plus

```

There is more information regarding the authentication process in `V$SESSION_CONNECT_INFO`; for example, the right-most column of Table 4.1 lists additional authentication information for my SQL*Plus session. Note that the authentication type is native (DATABASE):

Table 4.1 Contents of `V$SESSION_CONNECT_INFO` Matching the Logon Information in `V$SESSION`

SID	AUTHENTICATION _TYPE	OSUSER	NETWORK_SERVICE_BANNER
138	DATABASE	RON-SNYHR85G9DJ\ronb	Oracle Bequeath NT Protocol Adapter for 32-bit Windows: Version 10.1.0.2.0 – Production
138	DATABASE	RON-SNYHR85G9DJ\ronb	Oracle Advanced Security: authentica- tion service for 32-bit Windows: Version 10.1.0.2.0 – Production

Table 4.1 Contents of `V$SESSION_CONNECT_INFO` Matching the Logon Information in `V$SESSION`

SID	AUTHENTICATION _TYPE	OSUSER	NETWORK_SERVICE_BANNER
138	DATABASE	RON-SNYHR85G9DJ\ronb	Oracle Advanced Security: NTS authentication service adapter for 32-bit Windows: Version 2.0.0.0.0
138	DATABASE	RON-SNYHR85G9DJ\ronb	Oracle Advanced Security: encryption service for 32-bit Windows: Version 10.1.0.2.0 – Production
138	DATABASE	RON-SNYHR85G9DJ\ronb	Oracle Advanced Security: crypto-checksumming service for 32-bit Windows: Version 10.1.0.2.0 – Production

It turns out that on Windows, Oracle also suggests that you use operating system authentication as a best practice. When using operating system authentication, Oracle has several parameters you can use to fine-tune the authentication process. These are initially set up in `init.ora`, but you can look at the values by selecting from `V$PARAMETER` or by signing on to SQL*Plus and running `SHOW PARAMETERS`. This lists all of the current parameters. The following four parameters (with the default values in a 10g installation) are relevant in the context of using operating system authentication:

```

remote_os_authent          boolean    FALSE
remote_os_roles            boolean    FALSE
os_authent_prefix          string     OPSS$
os_roles                   boolean    FALSE

```

The first parameter—`remote_os_authent`—is equivalent to the `CLIENT` authentication for DB2, and you should always set it to `FALSE`. If set to `true`, it means that the server trusts that the client has authenticated the user on the remote operating system and does not require further authentication. In the same spirit, `remote_os_roles` should be set to `FALSE`, because this parameter allows a client authenticated remotely to enable operating system roles. The `os_authent_prefix` controls the mapping between operating system users on the server to database users. Users who have already been authenticated by the server's operating system can sign onto Oracle without entering a password. The question is how the usernames in both systems are related. This parameter is appended as a prefix to the username used by the operating system and is useful in situations where you may have the same usernames in the database as in the operat-

ing system but do not necessarily want them mapped to one another. For example, I can have an operating system user named Scott, and this is perhaps someone who never uses the database, so I therefore don't want this OS user to be able to automatically sign onto the database. This is why the default is not an empty string. In some cases, you may want to change this value to an empty string to simplify the mapping between users. Finally, `os_roles` allows you to control which roles are granted through the operating system rather than through the database and should be used when you want the operating system to control not only authentication but also parts of the authorization process.

Windows-based authentication in Oracle means that Oracle uses Windows API calls to verify the identity of the connection request. This only works when both the client and the server are running on Windows. You will also need to set the following in your `$ORACLE_HOME\network\admin\sqlnet.ora` (which is the default value when you install Oracle on Windows):

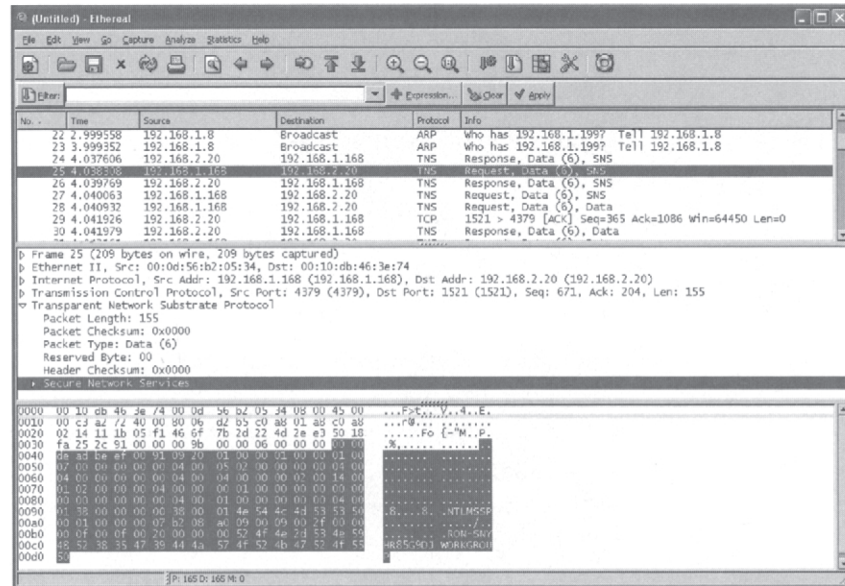
```
SQLNET.AUTHENTICATION_SERVICES=(NTS)
```

If you set this value, you are telling the Oracle server that it should first try to perform Windows authentication, and only if that is not possible it should fall back on native authentication.

Let's see what takes place when such a connection is attempted when starting up SQL*Plus on the client machine. In this case, you enter the username, password, and service name in the SQL*Plus sign-on screen. The TNS layer sees that you have NTS authentication configured on the client side (by looking at `sqlnet.ora`), and therefore the client sends a connection request to the server specifying that you would like to use NTS authentication. If the server is also configured to use Windows authentication, it will accept the request; the client and server have negotiated to use Windows authentication. You can actually see this action take place in the communication stream (for more on how to use packet sniffers and what these packet dumps mean, please see Chapter 10). For example, if you inspect the network conversations between two Windows machines, you will constantly see TNS packets marked as SNS (Secure Network Services), which is used in the authentication process within TNS. You can see an example in Figure 4.3 (Windows authentication elements are highlighted in all three panes):

If you were to look at an authentication process with your client connecting to a UNIX or Linux machine, some of these packets would be missing because the server would immediately answer that it cannot do

Figure 4.3
 Capture of the
 TNS connection
 setup process using
 Windows
 authentication.



Windows authentication. If you look inside the packet in the Windows-to-Windows scenario shown earlier, you can see that authentication is going to use NTLMSSP.

```

0000 00 10 db 46 3e 74 00 0d 56 b2 05 34 08 00 45 00 ...F>t.. V..4..E.
0010 01 07 a2 73 40 00 80 06 d2 70 c0 a8 01 a8 c0 a8 ...s@... .p.....
0020 02 14 11 1b 05 f1 46 6f 7b c8 22 4d 2f 84 50 18 .....Fo {."M/.P.
0030 f9 84 ce e5 00 00 00 df 00 00 06 00 00 00 00 00 .....
0040 de ad be ef 00 d5 09 20 01 00 00 01 00 00 01 00 .....
0050 02 00 00 00 00 00 04 00 01 b4 00 00 00 00 b4 00 .....
0060 01 4e 54 4c 4d 53 53 50 00 03 00 00 00 18 00 18 .NTLMSSP .....
0070 00 84 00 00 00 18 00 18 00 9c 00 00 00 1e 00 1e ..@..... ^.....
0080 00 40 00 00 00 08 00 08 00 5e 00 00 00 1e 00 1e ..f.....
0090 00 66 00 00 00 00 00 00 00 b4 00 00 00 05 82 88 ..f.....
00a0 a0 52 00 4f 00 4e 00 2d 00 53 00 4e 00 59 00 48 ..R.O.N.-.S.N.Y.H
00b0 00 52 00 38 00 35 00 47 00 39 00 44 00 4a 00 72 ..R.8.5.G.9.D.J.r
00c0 00 6f 00 6e 00 62 00 52 00 4f 00 4e 00 2d 00 53 ..o.n.b.R .O.N.-.S
00d0 00 4e 00 59 00 48 00 52 00 38 00 35 00 47 00 39 ..N.Y.H.R .8.5.G.9
00e0 00 44 00 4a 00 54 00 aa 32 42 2a ad 62 00 00 00 ..D.J.T.. 2B*.b...
00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 33 94 30 ..... 3.0
0100 d7 f5 c6 4a 5f 41 b9 aa 4b aa 31 35 df c5 25 9d ...J_A.. K.15..%.
0110 56 70 22 72 9d Vp"r.
  
```

NTLMSSP stands for the NTLM Security Support Provider, and NTLM stands for NT LAN Manager. NTLM is an authentication protocol used in various Microsoft network protocol implementations and supported by the NTLM Security Support Provider (NTLMSSP). Originally used for authentication and negotiation of secure DCE/RPC, NTLM is also used throughout Microsoft's systems as an integrated single sign-on

mechanism. NTLMSSP is common, but other mechanisms could be used—one good example being Kerberos.

At this point the client needs to send the entered credentials to the server, so the username and password are sent to the server. The password is not sent in the clear (some of the packet contents have been omitted). The actual password hashing mechanism is beyond the scope of this chapter; if you are interested in this detail, please refer to the *Oracle Security Handbook* by Theriault and Newman (McGraw-Hill, 2001).

```

0000 00 10 db 46 3e 74 00 0d 56 b2 05 34 08 00 45 00 ...F>t.. V..4..E.
0010 03 53 a2 77 40 00 80 06 d0 20 c0 a8 01 a8 c0 a8 .S.w@... ..
0020 02 14 11 1b 05 f1 46 6f 7d d7 22 4d 30 eb 50 18 .....Fo }. "MO.P.
0030 f8 1d 6f 52 00 00 03 2b 00 00 06 00 00 00 00 ..oR...+ .....
0040 03 73 03 0c a2 e1 00 05 00 00 00 01 01 00 00 6c .s..... ..l
0050 b0 12 00 07 00 00 00 24 ad 12 00 b0 b2 12 00 05 .....$ .....
0060 73 63 6f 74 74 0d 00 00 00 0d 41 55 54 48 5f 50 scott... ..AUTH_P
0070 41 53 53 57 4f 52 44 20 00 00 00 20 31 38 30 31 ASSWORD ... 1801
0080 36 43 31 31 37 32 35 46 44 38 37 32 30 36 42 30 6C11725F D87206B0
0090 44 37 36 42 32 37 37 30 31 43 42 44 00 00 00 00 D76B2770 1CBD....
00a0 0d 00 00 00 0d 41 55 54 48 5f 54 45 52 4d 49 4e .....AUT H_TERMIN
00b0 41 4c 0f 00 00 00 0f 52 4f 4e 2d 53 4e 59 48 52 AL.....R ON-SNYHR
00c0 38 35 47 39 44 4a 00 00 00 00 0f 00 00 00 0f 41 85G9DJ.. .....A
00d0 55 54 48 5f 50 52 4f 47 52 41 4d 5f 4e 4d 0c 00 UTH_PROG RAM_NM..
00e0 00 00 0c 73 71 6c 70 6c 75 73 77 2e 65 78 65 00 ...sqlpl usw.exe.
00f0 00 00 00 0c 00 00 00 0c 41 55 54 48 5f 4d 41 43 ..... AUTH_MAC
0100 48 49 4e 45 1a 00 00 00 1a 57 4f 52 4b 47 52 4f HINE.... .WORKGRO
0110 55 50 5c 52 4f 4e 2d 53 4e 59 48 52 38 35 47 39 UP\RON-S NYHR85G9
0120 44 4a 00 00 00 00 00 08 00 00 00 08 41 55 54 48 DJ..... ..AUTH
0130 5f 50 49 44 0b 00 00 00 0b 35 36 32 38 34 3a 35 _PID.... .56284:5
0140 36 32 38 38 00 00 00 00 08 00 00 00 08 41 55 54 6288.... ..AUT
0150 48 5f 41 43 4c 04 00 00 00 04 34 34 30 30 00 00 H_ACL... ..4400..
0160 00 00 12 00 00 00 12 41 55 54 48 5f 41 4c 54 45 .....A UTH_ALTE
0170 52 5f 53 45 53 49 4f 4e dc 01 00 00 fe ff 41 R_SESSIO N.....A

```

One word of caution regarding passwords in clear text: While the sign-on process does not transit passwords in clear text, changing a password usually does. This means that if someone is eavesdropping on your communications, they will be able to see passwords if they are changed. All databases that can manage passwords have this potential vulnerability. Here are two examples:

To change a password in SQL Server, you can execute `sp_password` giving the old password and the new password:

```

exec sp_password 'password', 'n3wp2ssw4rd'
go

```

Both passwords are sent in the clear over the network:

```

0000 00 10 db 46 3e 74 00 0d 56 b2 05 34 08 00 45 00 ...F>t.. V..4..E.
0010 00 88 be 9d 40 00 80 06 b6 c4 c0 a8 01 a8 c0 a8 .....@... ..
0020 02 15 10 0e 05 99 e8 2c d8 6d 8d 18 7b f3 50 18 ..... ,m..{.P.
0030 f6 46 25 5a 00 00 01 01 00 60 00 00 01 00 65 00 .F%Z.... .`.....e.

```

```

0040 78 00 65 00 63 00 20 00 73 00 70 00 5f 00 70 00 x.e.c. . s.p._.p.
0050 61 00 73 00 73 00 77 00 6f 00 72 00 64 00 20 00 a.s.s.w. o.r.d. .
0060 27 00 70 00 61 00 73 00 73 00 77 00 6f 00 72 00 '.p.a.s. s.w.o.r.
0070 64 00 27 00 2c 00 20 00 27 00 6e 00 33 00 77 00 d.',, . '.n.3.w.
0080 70 00 32 00 73 00 73 00 77 00 34 00 72 00 64 00 p.2.s.s. w.4.r.d.
0090 27 00 0d 00 0a 00 '.....

```

The same is true for Oracle; executing:

```
alter user scott identified by n3wp2ssw4rd;
```

generates the following network communication:

```

0000 00 10 db 46 3e 74 00 0d 56 b2 05 34 08 00 45 00 ...F>t.. V..4..E.
0010 00 ef d3 f4 40 00 80 06 a1 07 c0 a8 01 a8 c0 a8 ....@... .....
0020 02 14 11 fd 05 f1 f6 eb c8 8f 53 01 76 42 50 18 ..... ..S.vBP.
0030 f6 ba 2c 7c 00 00 00 c7 00 00 06 00 00 00 00 00 ..,|.....
0040 11 69 20 b0 3f e1 00 01 00 00 00 02 00 00 00 03 .i.?.?....
0050 5e 21 21 80 00 00 00 00 00 00 f0 99 e2 00 2a 00 ^!!..... ..*..
0060 00 00 d8 de e1 00 0c 00 00 00 00 00 00 00 08 df .....
0070 e1 00 00 00 00 00 01 00 00 00 00 00 00 00 00 .....
0080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0090 00 00 00 00 00 00 0a df e1 00 cc 9d e2 00 00 00 .....
00a0 00 00 2a 61 6c 74 65 72 20 75 73 65 72 20 73 63 ..*alter user sc
00b0 6f 74 74 20 69 64 65 6e 74 69 66 69 65 64 20 62 ott iden tified b
00c0 79 20 6e 33 77 70 32 73 73 77 34 72 64 01 00 00 y n3wp2s sw4rd...
00d0 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00e0 00 00 00 00 00 00 00 00 00 07 00 00 00 00 00 .....
00f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

Chapter 10 shows you how you can protect yourself from this type of vulnerability by encrypting the communications stream. Also, if you are using operating system authentication, you can avoid this database issue because the password change does not really occur by communicating with the database—it happens at the operating system level.

Let's go back to Windows authentication in Oracle. You now understand how the client connects to the server and how the server uses the Windows APIs for authentication. The next step in terms of the sign-on process is for the server to associate the authenticated user with an Oracle user. If I have an operating system user called ronb, for example, I would use:

```
CREATE USER "OPSS$RONB\WORKGROUP" IDENTIFIED EXTERNALLY;
```

IDENTIFIED EXTERNALLY tells Oracle that authentication is done outside the database, and that's why I don't need to specify a password when doing so. The OPSS\$ is the prefix defined by the `os_authent_prefix` attribute mentioned a few paragraphs ago. One of the advantages of this approach is that you would never change this user's passwords using ALTER USER—you would change the password in Windows.

Before moving on to the next topic, one last word on using the operating system for authentication. When the operating system provides authentication services, it may also be used to associate the user signing onto the database with groups. This is a part of the authorization layer and can have a broader impact on database security. For example, the same application may behave differently when accessing a database deployed on a UNIX system versus the same database deployed on a Windows system. Furthermore, any change to the user definitions at the operating system level may change not only whether the user can sign onto the database but also what they are entitled to do. This is a serious statement, and many people view this as giving away too much control.

An example of this behavior can occur in DB2 UDB on Windows. When you sign on using `SERVER` authentication, Windows not only handles authentication, but it also returns an access token that includes information about the groups the user belongs to, potentially including local groups, global groups, domain local groups, and universal groups. You can control the process of group lookup through the `DB2_GRP_LOOKUP` variable that can be set using the `db2set` utility. The values that this variable can take are as follows:

- `TOKEN`. Association is done based on the domain at which the user is defined.
- `TOKENLOCAL`. Association is done based on local groups at the database server.
- `TOKENDOMAIN`. Association is done based on all domain groups that the user belongs to.

When you use the `db2set` utility to set this configuration, you can use these three values to define either local groups, domain groups, or both:

```
db2set DB2_GRP_LOOKUP=LOCAL,TOKENLOCAL
db2set DB2_GRP_LOOKUP=DOMAIN,TOKENDOMAIN
db2set DB2_GRP_LOOKUP=,TOKEN
```

4.2 Understand who gets system administration privileges

At this point you may be confused, and you are probably not alone. The authentication and group association models become fairly complex, and the relationship between the operating system security model and the database security model do not make it easier. If you feel this way only because the concepts introduced in this chapter are new to you, then it's no big deal (maybe you need to read more on the subject and maybe you need to reread the previous sections). However, if you are confused about how all this is implemented within your own database environment, then you should put down this book and perform a comprehensive review of your security environment. There is absolutely nothing worse than a misunderstanding in this area.

When doing a review, it is helpful to follow these steps:

1. Review the authentication model
2. Review group association
3. Review role association
4. Review privilege association
5. Perform a “dry run”
6. Carefully inspect system administration privileges

Don't underestimate the benefit of item 5. When you go through an end-to-end process, you start to fully understand what is going on. In doing the dry run, you need to take a sample user trying to sign on to the database through the different layers and ask yourself to simulate what the OS and the database will do. You should do this process four times—once for a general user and once for an administrator user, each one both with local access and networked access.

Finally, in addition to paying special attention to administration users as part of the dry run, you should make sure you understand the effect the operating system can have on who gets system administration privileges to your database. For example, if you authenticate and associate groups through Windows, then any user account belonging to the local administrator group and potentially domain users belonging to the administrator

group at the domain controller will all have system administration privileges. Many people find this too risky, but the first step is to understand this.

4.3 Choose strong passwords

Passwords are your first line of defense and sometimes your only line of defense, so make sure you can count on them. Passwords are far too often left as defaults, often far too easy to guess, and often far too easy to crack. On the flip side, making sure you use strong passwords is probably one of the simplest things you can do and one of the best return-on-investments you can hope for.

4.3.1 Anatomy of the vulnerability: Guessing and cracking passwords

The simplest vulnerability in terms of weak passwords has to do with default and even empty passwords. While this seems trivial, you cannot begin to imagine the damage that this silly oversight has created and the cost to various IT organizations that can be directly attributed to empty passwords.

The most well-known vulnerability of this type involves Microsoft's SQL Server, and the attack is best known as the Spida worm or as SQL Snake. Spida came to the forefront in May 2002, when it attacked a large percentage of SQL Server systems all having an empty password for the `sa` account (the administrator account). See CERT incident note IN-2002-04 for more details (www.cert.org/incident_notes/IN-2002-04.html). The Spida worm scans for systems listening on port 1433 (the default port SQL Server listens on), and then tries to connect as the `sa` account using a null or simple password. If it gains access, it uses the `xp_cmdshel1` extended procedure to enable and set a password for the guest account. If successful, the worm then does the following:

1. Assigns the guest user to the local Administrator and Domain Admins groups
2. Copies itself to the Windows operating system
3. Disables the guest account
4. Sets the `sa` password to the same password as the guest account

5. Executes at the operating system level—and begins scanning for other systems to infect—thus propagating itself in an exponential manner as do most worms
6. Attempts to send a copy of the local password (SAM) database, network configuration information, and other SQL server configuration information to a fixed e-mail address (ixtld@postone.com) via e-mail

Through step 5 the worm propagated itself rather quickly through many corporate environments. The success of the infection is completely dependent on the use of an empty `sa` password. Given that this was one of the most successful worms of all time, you can understand how prevalent this bad practice was (and hopefully is no longer). In fact, while this is no longer true today, SQL Server used to ship with an empty `sa` password. It is therefore not too surprising that this worm was so successful, especially given that this vulnerability also exists in SQL Server's "baby brother" Microsoft Data Engine (MSDE), which runs embedded on so many workstations. Its success has earned it a "respectable" contribution to make SQL Server the fourth place in the SANS top 10 Windows vulnerabilities (see www.sans.org/top20 for more information).

Interestingly enough, Microsoft published an article more than six months before the eruption of Spida citing a new worm code-named "Voyager Alpha Force" that also uses a blank `sa` password. In Article 313418, Microsoft says:

A worm, code-named "Voyager Alpha Force," that takes advantage of blank SQL Server system administrator (`sa`) passwords has been found on the Internet. The worm looks for a server that is running SQL Server by scanning for port 1433. Port 1433 is the SQL Server default port. If the worm finds a server, it tries to log in to the default instance of that SQL Server with a blank (NULL) `sa` password.

If the login is successful, it broadcasts the address of the unprotected SQL Server on an Internet Relay Chat (IRC) channel, and then tries to load and run an executable file from an FTP site in the Philippines. Logging in to SQL Server as `sa` gives the user administrative access to the computer, and depending on your particular environment, possibly access to other computers.

It is unfortunate that this awareness did not help circumvent Spida. It is also unfortunate that in the same article Microsoft continues to say:

Important: There is no bug in SQL Server that permits this penetration; it is a vulnerability that is created by an unsecured system.

This may be important to Microsoft, but it certainly is not important to Microsoft's customers. Furthermore, one can claim that shipping with an empty password IS a bug and after Spida, Microsoft quickly changed the shipping password for sa, and today Microsoft is far more proactive in making sure that its customers are better protected even if it is "not a Microsoft bug."

Incidentally, weak default passwords also exist in other database products. Before version 9i R2, Oracle shipped with a password of `MANAGER` for the `SYSTEM` account and a password of `CHANGE_ON_INSTALL` for the `SYS` account—both accounts providing elevated privileges.

The next type of attack you should be aware of uses password crackers. These tools automate the process of signing onto your database and use a file of words to guess passwords. They iterate through all of the words in the files, and if your password is included in this list, they will eventually manage to sign onto the database.

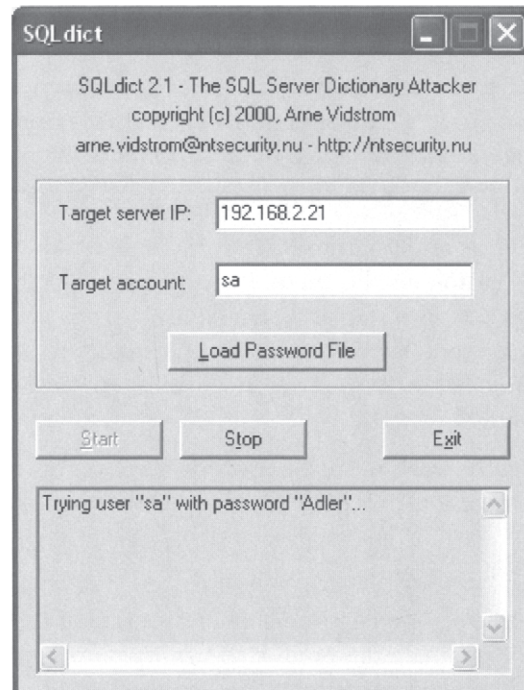
An example of such a tool is SQLdict, which you use to run a dictionary attack on a SQL Server instance; you can download the tool from www.ntsecurity.nu/toolbox/sqldict. To use it, you first need to get a password file—a great place for those is <ftp://ftp.ox.ac.uk/pub/wordlists/>. Once you have the file(s), open the tool, point it at the target SQL Server, enter the target account, load a password file, and click the Start button, as shown in Figure 4.4. If your password is in the dictionary file, it will eventually be cracked.

SQLdict is a simple tool that a hacker may use. As a DBA testing the strength of your passwords, you will typically use another form of tools mentioned in the next subsection.

4.3.2 Implementation options: Promote and verify the use of strong passwords

Resolving the issues detailed in the previous subsection is easy. Don't use empty passwords. Don't leave any default passwords. Audit your passwords. Use password best practices. Use a password cracker tool—after all, the

Figure 4.4
*Using SQLdict to
run a dictionary
attack on the sa
account in SQL
Server.*



hackers will probably try that as well. And finally, track for failed login attempts to alert you in case a password cracking tool is used.

Here are some simple dos and dont's:

Do:

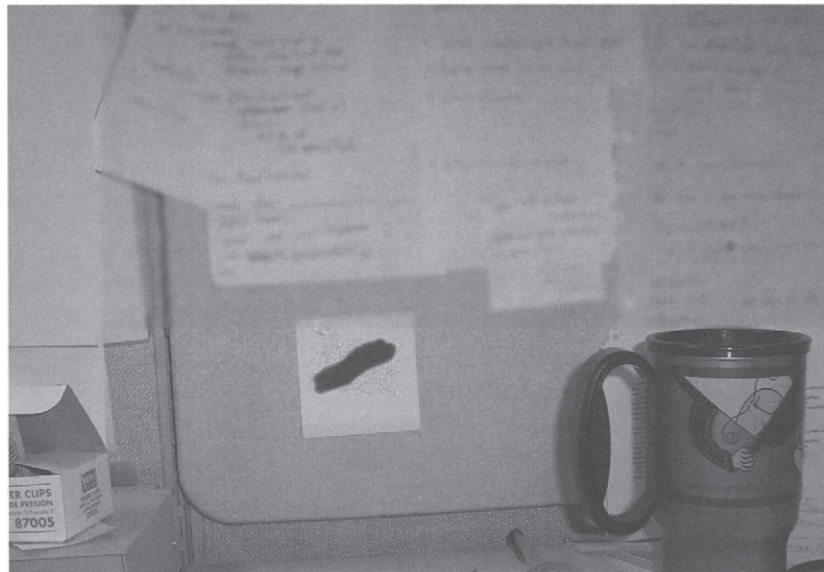
- Use a password with mixed-case letters.
- Use numbers in your passwords. I like the method that takes vowels and replaces them with numbers—it is good and easy to remember. For example, take a word such as *malicious* and replace vowels with numbers to get the password *m211c108s*. Don't use only this method, though, because a human hacker can try to guess at these if they see that you always use this method. Also, don't map the vowels to numbers always in the same way.
- Use punctuation marks within your passwords.
- Use passwords with at least six characters, and a minimum of eight is even better.
- If possible, choose a password that can be typed quickly and that cannot be easily guessed if someone looks over your shoulder.

Don't:

- Use the same password (even if it is strong) all over the place. At some point in time you will probably give it to someone, and if you use it in 50 different systems you have just given access to all 50 systems; you are also less likely to be willing to change all 50 passwords.
- Use the username as the password or any permutation of the login name (e.g., username spelled backward).
- Use words that can be looked up in a dictionary because they will appear in password cracker files.
- Use information that is easily obtained, such as your mother's maiden name, your children's names, or your pet's name.
- Use dates (such as your hiring date).

One other word of caution: you should strive for a strong password that you can remember. If you cannot remember your passwords, you will end up posting it on a sticky note or writing it down next to your computer, in which case you're back to square one. Figure 4.5 is a photo I took showing a “strong” database password that a developer found difficult to remember (which I blurred and marked out for obvious reasons). This, of course, never happens in your environment. 😊

Figure 4.5
*A good password
gone bad.*



Let's move on to password checking tools. You can use a tool such as SQLdict, but this is not very effective. It is slow and it creates a lot of "noise" (e.g., if you are alerting based on excessive failed logins, you will be spending the next five years deleting e-mails). From a performance standpoint, going through a dictionary with 100,000 words could take almost a full day. Instead, you can use a class of tools that run within the database and that use the fact that they have access to the database table where the password hashes are stored.

If you are running SQL Server, you can use the SQL Server Password Auditing Tool, which is available at www.cqure.net/tools.jsp?id=10. The tool assumes that you give it a text file with the usernames and password hashes as stored in the sysxlogins table. After downloading the tool, you should extract this information using:

```
select name, password from master..sysxlogins
```

and export it to a comma-delimited text file called hashes.txt. You then run the tool from the command line using:

```
sqlbf -u hashes.txt -d dictionary.dic -r out.rep
```

The tool is very fast. On my machine it made more than 200,000 guesses per second. You can also run a brute-force attack instead of a dictionary attack by running:

```
sqlbf -u hashes.txt -c default.cm -r out.rep
```

The `-c` flag tells the tool that the `.cm` file is a character set file. The default English file has the following character set, and you can change it if you have another locale:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

If you have an Oracle environment, you also have an abundance of tools. You can use any of the following tools to do password checking:

- Oracle Auditing Tools (OAT) is a set of tools that you can download from www.cqure.net/tools.jsp?id=7. Among the tools is OraclePWGuess, which is a dictionary attack tool.

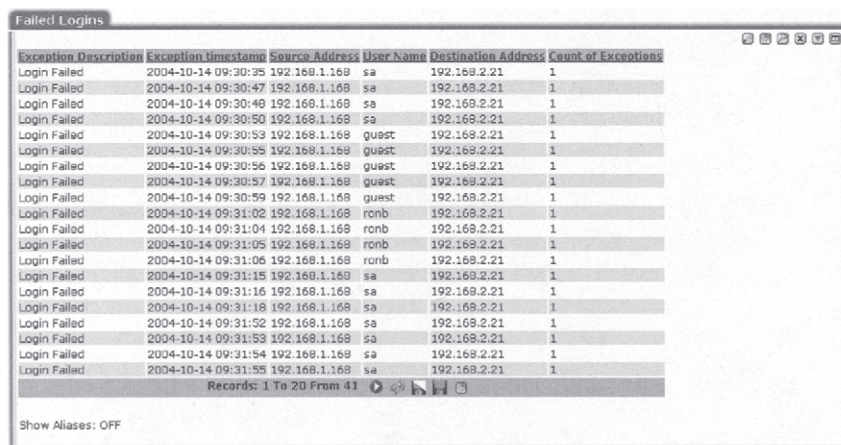
- Oracle Password Cracker by Adam Martin seems to be no longer available, but if you can find the download site, it is a nice tool to have.
- Oracle Password Cracker by Bead Dang is downloadable from www.petefinnigan.com/tool.htm and is a PL/SQL-based brute-force attack tool.

Note that these tools will only work if you are *not* using operating system authentication, because if you are using operating system authentication, the passwords are stored by the operating system and not the database. In this case you can use operating system-level password checkers (e.g., the Ripper password cracker: www.openwall.com/john).

Finally, you should always monitor failed login errors issued by the database. In a well-running environment, failed logins should occur only as a result of typing errors by users. Applications never have failed logins because they use coded usernames and passwords, and while users who log in using tools can have typing errors, these are the exception rather than the norm. By monitoring failed logins, you can easily identify suspicious behavior that may be a password attack.

When monitoring failed logins, you first need a simple way to create a list of all such occurrences. Once you have this information, you have two choices: you can either periodically look at a report that shows you all failed logins (as shown in Figure 4.6), or you can use this information to alert you when the number of failed logins goes over a certain threshold (as shown in

Figure 4.6
Report showing
failed login
information.



Exception Description	Exception timestamp	Source Address	User Name	Destination Address	Count of Exceptions
Login Failed	2004-10-14 09:30:35	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:30:47	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:30:48	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:30:50	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:30:53	192.168.1.168	guest	192.168.2.21	1
Login Failed	2004-10-14 09:30:55	192.168.1.168	guest	192.168.2.21	1
Login Failed	2004-10-14 09:30:56	192.168.1.168	guest	192.168.2.21	1
Login Failed	2004-10-14 09:30:57	192.168.1.168	guest	192.168.2.21	1
Login Failed	2004-10-14 09:30:59	192.168.1.168	guest	192.168.2.21	1
Login Failed	2004-10-14 09:31:02	192.168.1.168	ronb	192.168.2.21	1
Login Failed	2004-10-14 09:31:04	192.168.1.168	ronb	192.168.2.21	1
Login Failed	2004-10-14 09:31:05	192.168.1.168	ronb	192.168.2.21	1
Login Failed	2004-10-14 09:31:06	192.168.1.168	ronb	192.168.2.21	1
Login Failed	2004-10-14 09:31:15	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:31:16	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:31:18	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:31:52	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:31:53	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:31:54	192.168.1.168	sa	192.168.2.21	1
Login Failed	2004-10-14 09:31:55	192.168.1.168	sa	192.168.2.21	1

Records: 1 To 20 From 41

Show Aliases: OFF

Figure 4.7
 Creating an alert
 that sends an e-
 mail when failed
 logins go over a
 threshold of five.

Modify Alert

Name

Description

Run Frequency (minutes)

Active

Alert Definition

Query

Accumulation Interval (minutes)

Alert Threshold

Threshold per report per line

As absolute limit

As percentage change within period:

From To

Alert when value is threshold

Notification

Notification Frequency (minutes)

Notify as Urgent

Alert Receivers

MAIL	dba dba	<input type="button" value="Remove"/>
------	---------	---------------------------------------

Figure 4.7). Figure 4.7 shows a definition of an alert that is based on counting failed login events (the query) over a period of one hour and sending an e-mail notification to the DBA (the alert receiver) whenever the number of failed logins goes over a defined threshold (in this case the number is five). Regardless of whether you want active notification or whether you'll just periodically look at a report, you need a simple way to monitor these events. This can be done inside the database using native audit or trace features or by using an external security monitor that looks at all SQL calls and status codes communicated between clients and servers.

4.4 Implement account lockout after failed login attempts

In order to combat login attempts that are performed by hackers or people who do not own the account, you can choose to disable or lock out an account after a certain number of failed login attempts. This is especially useful to alleviate false logins by someone who watches over your shoulder when you type in your password and manages to get most of it but perhaps not all of it.

Account lockout can sometimes be implemented by the database (if the vendor supports it) and can always be implemented by an external security system. An example for doing this within the database is Oracle's support of the `FAILED_LOGIN_ATTEMPTS` attribute. Oracle can define security profiles (more on this in the next section) and associate them with users. In Oracle, one of the items you can set in a profile is the number of failed logins. In addition, you can set the number of days that the account will be locked out once the threshold for failed logins is exceeded. For example, to lock out Scott's account for two days in case of five failed login attempts, do:

```
SQL> CREATE PROFILE SECURE_PROFILE LIMIT
      2 FAILED_LOGIN_ATTEMPTS 5;
```

Profile created.

```
SQL> ALTER PROFILE SECURE_PROFILE LIMIT
      2 PASSWORD_LOCK_TIME 2;
```

Profile altered.

At this point you can look at your profile by running:

```
SELECT RESOURCE_NAME, LIMIT
       FROM DBA_PROFILES
       WHERE PROFILE='SECURE_PROFILE'
```

RESOURCE_NAME	LIMIT
COMPOSITE_LIMIT	DEFAULT
SESSIONS_PER_USER	DEFAULT
CPU_PER_SESSION	DEFAULT
CPU_PER_CALL	DEFAULT

LOGICAL_READS_PER_SESSION	DEFAULT
LOGICAL_READS_PER_CALL	DEFAULT
IDLE_TIME	DEFAULT
CONNECT_TIME	DEFAULT
PRIVATE_SGA	DEFAULT
FAILED_LOGIN_ATTEMPTS	5
PASSWORD_LIFE_TIME	DEFAULT
PASSWORD_REUSE_TIME	DEFAULT
PASSWORD_REUSE_MAX	DEFAULT
PASSWORD_VERIFY_FUNCTION	DEFAULT
PASSWORD_LOCK_TIME	2
PASSWORD_GRACE_TIME	DEFAULT

Finally, associate the profile with the user:

```
ALTER USER SCOTT PROFILE SECURE_PROFILE;
```

If your database does not support this function, you can use an external security system, as shown in Figure 4.7. You can cause a database operation to be invoked rather than a notification. Following the example of the previous section, instead of sending a notification to the DBA that the threshold is exceeded, you can configure the alert to sign onto the database server using an administrator account and lock out an account using built-in stored procedures. For example, if you are running a Sybase ASE server, the external system can call the `sp_locklogin` procedure.

4.4.1 Anatomy of a related vulnerability: Possible denial-of-service attack

One thing you should realize when implementing account lockout after a certain number of failed logins is that it can be used against you, in the form of a denial-of-service attack (DoS attack). A DoS attack is one where the attacker does not manage to compromise a service, gain elevated privileges, or steal information. Instead, he or she brings the service down or cripples it to a point that legitimate users of the service cannot use it effectively. This is the hacker's equivalent of vandalism.

If you implement account lockout after five failed login attempts to a certain account within an hour, a hacker can create a DoS attack based on trying to sign on to the database using legitimate usernames and bad passwords. Any password will do, because the attack is simply based on the fact that if I have a list of usernames (or can guess them), then I can quickly

cause every single one of these accounts to be locked out within a matter of minutes (even with a simple tool such as SQLdict).

4.4.2 Implementation options for DoS vulnerability: Denying a connection instead of account lockout

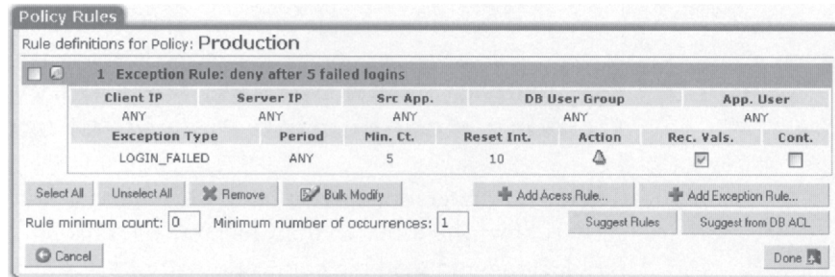
There is an inherent problem here: the DoS attack uses precisely the same scenario for which the account lockout was created. You can achieve a lot by blocking and denying connection attempts rather than locking out an account, especially if you can block a connection based on many parameters rather than just the login name. This can usually only be done using an external security system such as a database firewall. In this case a failed login event has additional qualifiers other than the login name, such as the IP address from which the request is coming. For example, the denial rule shown in Figure 4.8 will deny all access after five failed login attempts, but will do so only to requests coming from the client IP address and going to the server IP address on which the failed login attempts occurred. In this scenario, a hacker who tries to mount a DoS attack will only succeed in making sure that all connection attempts from his/her workstation are denied but will not cause any harm to legitimate users (and their workstations).

4.5 Create and enforce password profiles

Continuing with the example profile from the previous section, some databases allow you to enforce good password management practices using password profiles. You already saw how Oracle uses profiles to enforce account lockout, but you can set additional limits per profile:

- `PASSWORD_LIFE_TIME`. Limits the number of days the same password can be used for authentication
- `PASSWORD_REUSE_TIME`. Number of days before a password can be reused
- `PASSWORD_REUSE_MAX`. Number of password changes required before the current password can be reused
- `PASSWORD_GRACE_TIME`. Number of days after the grace period begins during which a warning is issued and login is allowed
- `PASSWORD_VERIFY_FUNCTION`. Password complexity verification script

Figure 4.8
Denial rule in
database firewall to
shut down
connections based
on failed logins.



Although Oracle is on of the most advanced in terms of setting such profiles, many of these functions exist in other databases as well. For example, Sybase ASE 12.5 allows you to require the use of digits in all passwords:

```
exec sp_configure "check password for digit", 1
```

4.6 Use passwords for all database components

Your database may have components of which you may not even be aware, and those components may need to be password-protected. Examples include embedded HTTP servers or even application servers that are sometimes bundled with the latest versions. These certainly must be secured with passwords, but even the core database engine often has such components. Therefore, it is critical that you review the architecture of your database server, understand the different components that are deployed, and make sure you use passwords to secure them.

4.6.1 Anatomy of the vulnerability: Hijacking the Oracle listener

Let's look at an example from the Oracle world. In the previous chapter you saw various vulnerabilities that exist within the Oracle listener—let's look at another issue. Default Oracle installations do not set a password on the listener, and many people don't even know that this is supported or that it is needed. This creates a set of serious vulnerabilities, all of which can be avoided by setting a strong password for the listener (in addition and unrelated to the passwords set for user accounts).

The Oracle installation comes with a utility called `lsnrctl`. This utility is used to configure the listener and can be used to configure a remote lis-

tener. If I'm a hacker I can install Oracle on my laptop and use the utility to connect to a remote listener. All I need to do is update `listener.ora` on my machine to include an alias for the remote server, and then I can fire up the `lsnrctl` utility. If the remote listener is not protected with a password, I can connect to it remotely!

Once I'm connected to a remote listener, I can do the following damage:

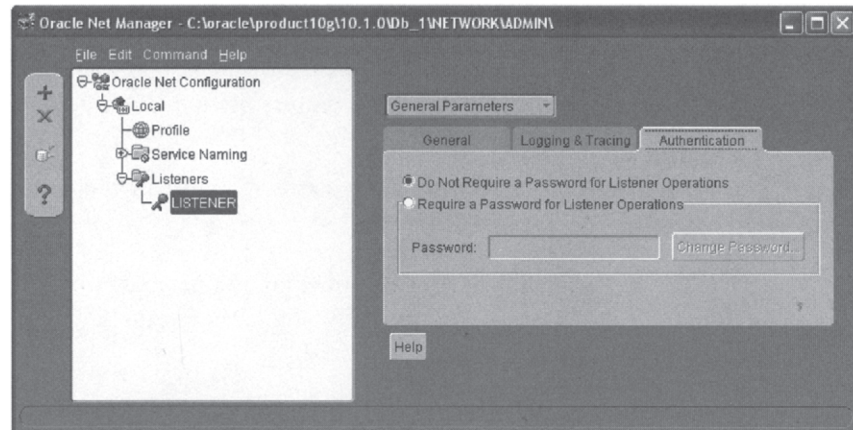
- I can stop the listener, making the database unreachable for any networked application. This in effect means I can bring the database down.
- I can get at information that is available to the listener, which will help me in hacking other parts of the database.
- I can write trace and log files that can impact the database or even the operating system.

The first attack type is self-explanatory and serious. I can even write a tiny script that runs in a loop and tries to connect to the remote listener every second. If it sees an active listener, it can then proceed to stop it. This can drive a DBA crazy because it seems like the listener can never start up. I can mix this up with another `lsnrctl` command—`set startup_waittime`—that causes the listener to wait before it starts up. In this case my script will certainly stop the listener before it has had a chance to start.

The second vulnerability is based on the fact that the listener can tell me many things about the system. For example, if I run the `services` command, I can learn of the services running on the server, including path and environment variables.

The third vulnerability is based on the fact that I can cause log files to be written to disk in any location open to the operating system user with which Oracle was installed. I can initiate traces that would be placed in directories that I could access. I can write to any location to which the Oracle user has permissions and can even overwrite files that affect the database's operations and even the Oracle account (e.g., `.rhosts`, `.cshrc`, `.profile`) on UNIX. I can place files under the root of a Web server and then download the file using a browser. Because the trace files are detailed, they can be used to steal information or mount an additional attack on the database.

Figure 4.9
Using the Oracle
Net manager to set
the listener
password.



4.6.2 Implementation options: Set the listener password

You should always set a password for your listener. This is easy and simple, and you should do it using the `lsnrctl` utility or the Oracle Net Manager in versions 9i and 10g (you can also do it by modifying the `listener.ora` file, but in this case the password will be in clear text). To change the password in `lsnrctl`, use the `change_password` command. To set it, use the `set_password` command. Then save the change using `save_config`. To set the password using the Oracle Net Manager, open the Oracle Net Configuration->Local->Listeners folder and select the appropriate listener from the tree view on the left. Then select General Parameters from the pulldown menu as shown in Figure 4.9. Click on the Require a Password for Listener Operations radio button and enter your password.

In the general case, you must understand the various services you are running and make sure they are all protected with a password.

4.7 Understand and secure authentication back doors

Although security is always of the utmost importance, protecting you from shooting yourself in the foot is also something that many database vendors care about. As such, there are often hidden back doors that are placed to allow you to recover from really bad mistakes. You should read up on these and make sure you take extra steps to secure them so they are not used as a starting point of an attack.

Let's look at an example from the world of DB2 UDB authentication. This particular back door was introduced for cases in which you inadvertently lock yourself when changing authentication configurations, especially when you are modifying the configuration file. Because the configuration file is protected by information in the configuration file (no, this is not a grammatical error), some errors could leave you out—permanently.

And so while back doors are a big no-no in the security world, being locked out of your own database forever is probably worse, and IBM chose to put in a special back door. This back door is available on all platforms DB2 UDB runs on, and it is based on a highly privileged local operating system security user that *always* has the privilege to update the database manager configuration file. In UNIX platforms this is the user owning the instance, and in Windows it is anyone who belongs to the local administrators group.

All vendors have such hidden back doors, and you need to know about them. You should assume that hackers certainly know about them, and so you should know what they are, what limitations they have, and what additional security measures you should take to secure them. For example, the DB2 UDB back door described previously is limited to local access—it cannot be used from a remote client. You can therefore introduce additional security provision at the local OS level or even physical security for console access.

4.8 Summary

In this chapter you learned about various best practices involving authentication and user account management. You saw that most database environments have various authentication options and that some of these can be sophisticated (and unfortunately, complex). You also saw that all of the vendors can support an authentication model that relies on the operating system for authentication and group association. Moreover, all of these vendors actually recommend this type of authentication model as the stronger authentication option.

After learning about authentication options, you also learned about password strength and password profiles as well as what user account/password maintenance you may want to do continuously. The issue of passwords will come up again in the next chapter, this time from a standpoint of serious vulnerabilities that occur when applications do not appropriately protect usernames and passwords that can be used to access the database. This discussion is part of a broader discussion of how application security affects database security, which is the topic of the next chapter.

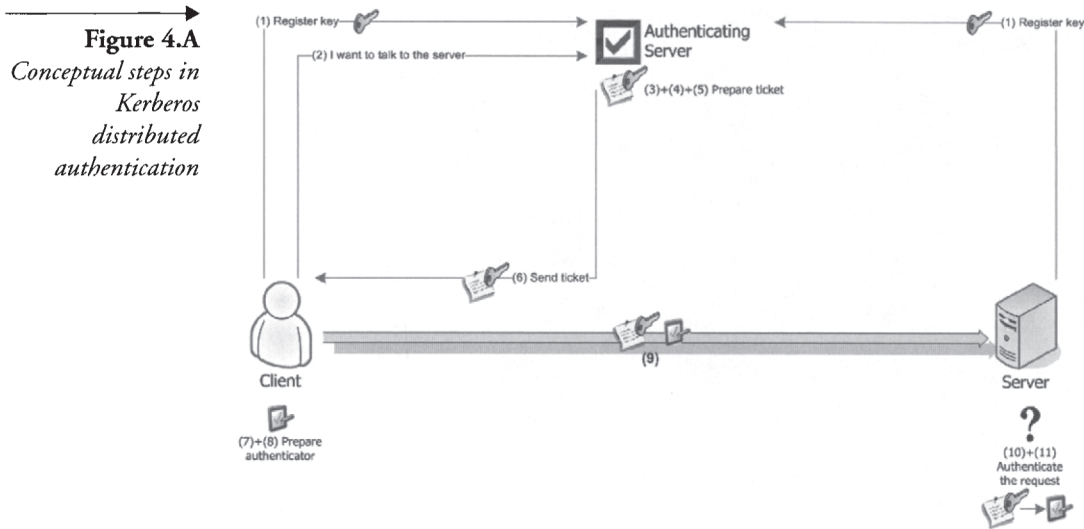
4.A A brief account of Kerberos

Kerberos is a distributed authentication system developed and distributed freely by the Massachusetts Institute of Technology (MIT). It has become a popular authentication mechanism and can be found in many environments. Whereas most authentication systems are based on the server requiring the client to present a password that is stored somewhere on the server, Kerberos asserts that the communication of the password makes the authentication scheme insecure and prone to an attack. The main principle implemented by Kerberos is the fact that the client can demonstrate to the server that it has some secret information (i.e., the password) without divulging the information. Instead, it relies on authenticating tickets.

Tickets are issued by a Kerberos Authentication Server (AS). In order to work with Kerberos, both the server and the client need to be registered with the AS and need to have encryption keys registered with the AS (step 1 in Figure 4.A). When a client wants to talk to a server, it communicates with the AS and sends it a request of the form “client A wants to talk to server B” (step 2). When the AS receives this request, it makes a brand-new encryption key called the *session key* (step 3). It takes the session key along with the name “server B” and encrypts it using the client’s key (step 4). It then takes the session key along with the name “client A” and encrypts it using the server’s key (step 5); this is called the *ticket*. Note that all of this is only possible because in step 1 the AS registers and maintains both keys.

Both the ticket and the encrypted session key are returned to the client (step 6). The client takes the first encrypted package and decrypts it using its key, allowing it to extract the session key (step 7) and check that the name “server B” is in there (avoiding man-in-the-middle replay attacks). The client then takes the ticket along with the current time and encrypts this combination using the session key (step 8). This package is called the *authenticator*. A timestamp is used to avoid replay attacks given that every ticket has a limited time frame within which it can be used. The client then communicates the ticket and the authenticator with the server (step 9).

When the server gets the ticket and the authenticator, it first uses its key to decrypt the ticket and extracts the session key and the name “client A” (step 10). It then uses the session key to decrypt the authenticator (step 11) and extract the timestamp and the ticket. If the timestamp differs from the server’s clock by too much, it rejects the request (note that Kerberos requires some form of clock synchronization between the entities). If the decrypted ticket matched the ticket received from the client, the server authenticated the request as coming from “client A” and can pass this to the



authorization layer. Notice that the client did not pass the password (or its key) to the server at any point in time.

In reality, Kerberos authentication is more complex than the flow shown in Figure 4.A. For example, in addition to the AS, Kerberos uses another server called the Ticket Granting Server (TGS), which together with the AS are called the Key Distribution Center (KDC). When a client wants to connect to a server, it first connects to the AS and requests to be authenticated with the TGS. It gets a ticket from the TGS (called the Ticket Granting Ticket, TGT). Every time the client wants to connect to a server, it requests a ticket from the TGS (and not the AS), and the reply from the TGS is not encrypted using the client's key but rather using the session key inside the TGT. I did not show this step in the flow shown in Figure 4.A, and Kerberos flows can be even more complex (e.g., in the context of cross-realm authentication), but all this is beyond the scope of this book.

Implementing Database Security and Auditing

A Guide for DBAs, information security administrators and auditors

Ron Ben Natan

Today, databases house our "information crown jewels", but database security is one of the weakest areas of most information security programs. With this excellent book, Ben-Natan empowers you to close this database security gap and raise your database security bar!

—**Bruce W. Moulton**, CISO/VP, Fidelity Investments (1995 - 2001)

It's been said that everyone has their 15 minutes of fame. You certainly don't want to gain yours by allowing a security breach in your database environment or being the unfortunate victim of one. Information and Data are the currency of On Demand computing, and protecting their integrity and security has never been more important. Ron's book should be compulsory reading for managing and maintaining a secure database environment.

—**Bob Picciano**, VP Database Servers, IBM

Let's start with a simple truth about today's world: If you have a database and you make it available to customers, employees, or whomever over a network, that database will be attacked by hackers—probably sooner rather than later. If you are responsible for that database's security, then you need to read this book. No other single source covers all of the many disciplines and layers involved in protecting exposed databases, and it especially shines in synthesizing all of its concepts and strategies into very practical and specific checklists of things you need to do. I've been an Oracle DBA for 15 years, but I'm not embarrassed to admit that five minutes into Chapter One I was making notes on simple measures I had overlooked.

—**Charles McClain**, Senior Oracle DBA
North River Consulting, Inc.

This book is about database security and auditing. You will learn many methods and techniques that will be helpful in securing, monitoring and auditing database environments. The book covers diverse topics that include all aspects of database security and auditing - including network security for databases, authentication and authorization issues, links and replication, database Trojans, etc. You will also learn of vulnerabilities and attacks



books.elsevier.com/digitalpress

that exist within various database environments or that have been used to attack databases (and that have since been fixed). These will often be explained to an "internals" level. There are many sections which outline the "anatomy of an attack" before delving into the details of how to combat such an attack. Equally important, you will learn about the database auditing landscape—both from a business and regulatory requirements perspective as well as from a technical implementation perspective.

- Useful to the database administrator and/or security administrator—regardless of the precise database vendor (or vendors) that you are using within your organization
- Has a large number of examples—examples that pertain to Oracle, SQL Server, DB2, Sybase and even MySQL..
- Many of the techniques you will see in this book will never be described in a manual or a book that is devoted to a certain database product
- Addressing complex issues must take into account more than just the database and focusing on capabilities that are provided only by the database vendor is not always enough. This book offers a broader view of the database environment— which is not dependent on the database platform—a view that is important to ensure good database security

Ron Ben Natan is CTO at Guardium Inc., a leader in database security and auditing. Prior to Guardium Ron worked for companies such as Intel, AT&T Bell Laboratories, Merrill Lynch, J.P. Morgan and ViryaNet. He holds a Ph.D. in the field of distributed computing from the University of Jerusalem. Ron is an expert on the subject of distributed application environments, application security and database security and has authored nine technical books and numerous articles on these topics.

Audience: DBAs, System and Network Administrators and Auditors

ISBN: 1-55558-334-2



Compliments of:



For more information contact:

IBM InfoSphere Guardium

5 Technology Park Drive guardium@us.ibm.com
Westford MA 01886 ibm.com/software/data/guardium