SECURITY

ELSEVIER
DIGITAL
PRESS

Ron Ben Natan

Implementing Database Security and Auditing

Includes Examples For:
ORACLE
SQL SERVER
DB2 UDB
SYBASE

1 Getting Started

This book is about database security and auditing. By reading it 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, and more. You will also learn of vulnerabilities and attacks 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. Many sections 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.

This book is written in a way that will be useful to you—the database administrator and/or security administrator—regardless of the precise database vendor (or vendors) that you are using within your organization. This is not to say that the book is theoretical. It is a practical handbook that describes issues you should address when implementing database security and auditing. As such, it has many examples that pertain to Oracle, SQL Server, DB2, Sybase, and sometimes even MySQL. However, because detailing every single example for every database platform would have meant a 2,000-page book, many of the examples are given for a single database or a couple of them. The good news is that all techniques (or almost all of them) are relevant to all database platforms, and I urge you to read through all sections even if the example code snippets are taken from a database environment that you are not running. In all of these cases, it will be easy for you to identify the equivalent setting or procedure within your own environment.

More important, 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. As you'll learn throughout this book, good database security cannot always be implemented solely within the database, and many of the most serious security issues that you may face as the database owner (or the server owner) have to do with the way applications use a database and the way various interacting systems are configured. Addressing these 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.

At this point you may be asking yourself a few questions:

- Doesn't the database have many security and auditing features? Isn't a database merely a file system with a set of value-added services such as transaction management and *security*? Isn't my database secure?
- Why now? The database has been part of the IT environment for many years (relational databases for at least 20 years); why should we suddenly be overly concerned with security and auditing?

The answer to the first set of questions is that while such features exist, they are not always used and are not always used correctly. Security issues are often a matter of misconfiguration, and the fact that the database implements a rich security model does not mean that it is being used or that it is being used correctly. If you are like 90% of database administrators or security administrators, you are probably aware that your database has big gaping holes—disasters waiting to happen. In fact, here are some examples that made the headlines (and rest assured that for every incident that makes headlines there are 100 that are kept quiet):

- In early 2000, the online music retailer CD Universe was compromised by a hacker known as "Maxus." The hacker stole credit card numbers from the retailer's database and tried to extort money from the retailer. When his demands were refused, he posted thousands of customers' credit card details to the Internet. (Go to http://databases.about.com/gi/dynamic/offsite.htm?site=http://www.pc%2Dradio.com/maxus.htm to see what Maxus' Web site looked like.)

- In December 2000, the online retailer Egghead.com announced that its customer database may have been compromised and warned that more than 3.5 million credit card numbers may have been stolen. Egghead.com later announced that the credit cards were not compromised but the investigation cost millions and few customers were willing to continue to do business with the retailer. The company went out of business shortly thereafter.
- In 2001, Bibliofind, a division of Amazon.com that specialized in rare and out-of-print books, was attacked and details for almost 100,000 credit cards were stolen. Even worse, the attackers maintained free access to the database for four months before being discovered! As a result, Bibliofind stopped offering buy/sell services and ended up as a matching service only (i.e., had to forgo a large portion of its revenues).
- In March 2001, the FBI reported that almost 50 bank and retail Web sites were attacked and compromised by Russian and Ukrainian hackers.
- In November 2001, Playboy.com was attacked and credit card information was stolen. In fact, the hackers sent e-mails to customers that displayed the credit card information.
- In the course of 2001, Indiana University was successfully attacked twice and private information, such as social security numbers and addresses, was stolen.
- A study conducted by Evans Data (a market research firm) in 2002 sampled 750 companies and reported that 10% of databases had a security incident in 2001! More than 40% of banking and financial services companies reported "incidents of unauthorized access and data corruption" and 18% of medical/healthcare firms reported similar types of incidents.
- In Oct. 2004 a hacker compromised a database containing sensitive information on more than 1.4 million California residents. The breach occurred on Aug 1 but was not detected until the end of the month. The database in question contained the names, addresses, Social Security numbers, and dates of birth of caregivers and care recipients participating in California's In-Home Supportive Services (IHSS) program since 2001. The data was being used in a UC Berkeley study of the effect of wages on in-home care and was obtained with authorization from the California Department of Social Services. The hacker had reportedly taken advantage of an unpatched system and

while officials declined to state which vendor's database was the subject of the attack they did report that it was a "commercially available product with a known vulnerability that was exploited."

- In Jan 2005 the following was reported by Security Focus (http://www.securityfocus.com/news/10271):

  A sophisticated computer hacker had access to servers at wireless giant T-Mobile for at least a year, which he used to monitor U.S. Secret Service e-mail, obtain customers' passwords and Social Security numbers, and download candid photos taken by Sidekick users, including Hollywood celebrities, SecurityFocus has learned… by late July [of 2004] the company had confirmed that the offer was genuine: a hacker had indeed breached their customer database

The answer to the second set of questions—why now?—is a convergence of several factors—almost a "perfect storm." True, the database has been around for a long time, but the following trends are dominating the last few years:

- E-commerce and e-business
- New and wonderful ways to use databases
- Increased awareness among the hacker community
- Widespread regulations that pertain to IT and to security

E-commerce and e-business have changed the way we live. We buy from online retailers, we pay our utility bills using online banking sites, and more. Businesses have optimized their supply chains and use Customer Relationship Management (CRM) software to manage relationships with their clients. In doing so, systems have become much "closer" to each other and much "closer" to the end users. Sure, we use firewalls to secure our networks and we don't connect databases directly to the Internet, but you'll see in Chapter 5 that there is more than one way to skin a cat and that databases are far more exposed than they used to be. Ten years ago the database was accessed by applications that were only available to internal employees. Now it is (indirectly through the application) accessed by anyone who has access to the Web site (i.e., everyone in the world).

While e-commerce has certainly added many indirect users on the database, e-business has had a much bigger impact on security (or the lack of it). Doing efficient business with suppliers, customers, and employees has created new and wonderful ways in which the database is used and innovative ways in which it is configured. Opening up the enterprise to improve processes and streamline business was done quickly and without too much analysis of security implications. Databases are deployed in many places (physically and logically) and often with no significant protective layers.

New technologies are constantly being released by the vendors. These technologies include Web services within the database, XML handling within the database, tight integration with application servers, and the ability to run any application logic directly within the database (to the extent of having an embedded Java virtual machine inside the database). This is great for developers and for increasing productivity, but it creates a security nightmare. More functionality means more (actually, many more) bugs that can be exploited by hackers, and many of the leading vendor databases have been plagued with bug-related vulnerabilities. Even if new functions have no vulnerability, these features are usually risky because they open up the database to more types of attacks. They increase not only the developer's productivity but also the hacker's productivity.

While we're discussing hacker skills and effectiveness, let's move on to hacker awareness. Hackers are always looking for new targets for their attacks and new methods they can use. In the same way that you realize that databases hold the crown jewels, so do the hackers. Furthermore, after mastering attacks on networks and operating systems, hackers have turned to applications and databases as new breeding ground. This is very visible in hacker forums. It is interesting, for example, to track hacker conferences such as BlackHat and Defcon. In 2001, both BlackHat and Defcon had one presentation each devoted to database hacking. In 2002, BlackHat had five such presentations and Defcon had four such presentations. In 2003, BlackHat already had a full track dedicated to database hacking.

Last, but by no means least, is regulation. Bad accounting practices, fraud, and various corporate scandals/crimes have prompted regulators to define and enforce new regulations that have a direct impact on IT auditing. Because financial, personal, and sensitive data is stored within databases, these requirements usually imply database auditing requirements. Because regulations such as Sarbanes-Oxley, GLBA, and HIPAA (all discussed in Chapter 11) have financial and criminal penalties associated with noncompliance, database security and auditing have suddenly come to the forefront.

So now that you are (hopefully) convinced that you need to invest in the security of your database, let's turn to the book. The book has two main parts: Chapters 1 through 10 show you how to implement various facets of database security, and Chapters 11 through 13 can help you with database auditing implementations. Each chapter is focused on a certain aspect of the database. For example, Chapter 3 is focused on the database as a networked server, Chapter 4 on database authentication, and Chapter 10 on encryption within the database environment. The only exception is this chapter—Chapter 1. In this chapter you will get started by taking care of the basics—various best practices in terms of hardening your database, applying patches, and so on. This is also the most boring chapter of the book, specifically because it includes long lists of things you should remember when starting off. Don't skip this chapter, because it has many useful snippets of experience, but remember that the rest of the book is much more elaborate and much more annotated than this chapter.

1.1 Harden your database environment

Hardening is a process by which you make your database more secure and is sometimes referred to as locking down the database. When you harden your database environment, you remove vulnerabilities that result from lax configuration options and can even compensate for vulnerabilities that are caused by vendor bugs. Although you cannot remediate these bugs, you can form an environment in which those bugs cannot be exploited.

Hardening is also called hack-proofing. The essence of the process involves three main principles. The first involves locking down access to important resources that can be misused—maliciously or by mistake. The second involves disabling functions that are not required for your implementation, which can be misused by their very existence. The third principle is that of least privileges (i.e., giving every user, task, and process the minimal set of privileges required to fulfill their role).

Hardening is a process that is relevant to any resource within IT, and hardening scripts are available for every operating system, server, and so on. In many ways you can view the entire book as a hardening guide; in each chapter you will focus on one aspect of the relational database management system (RDBMS), learn how it can be misused, and what you should do to avoid these cases. The lists presented below do not go into that level of detail and do not cover the many dimensions of database security that are covered by Chapters 3 through 10. Instead, this section provides a starting point after which the lessons learned in later chapters can be implemented.

This section is broken up into different database types, but many of the tasks are common and do not depend on the particular database platform. For example, good security always starts with securing the physical environment and the operating system (OS) the database runs on and ends with disallowing developer access to production instances. Apart from mentioning these as list items, I do not go into the details of how to secure the OS layer because there are many books on that topic alone. (see the resource section at the end of this chapter)

1.1.1 Hardening an Oracle environment

Oracle is one of the most well-documented database environments, and there are many hardening scripts on the Web (e.g., Pete Finnigan's checklist at www.petefinnigan.com/orasec.htm). Hardening an Oracle environment should include at least the following tasks:

- Physically secure the server on which Oracle lives.
- In a UNIX environment:
  - Don't install Oracle as root.
  - Before installing, set the umask to 022.
  - Don't use /tmp as the temporary install directory; use a directory with 700 permissions.
- In a Windows environment, do not install Oracle on a domain controller.
- Create an account for each DBA that will access the server; don't have all DBAs logging into the server using the the same user.
- Lock the software owner account; don't use it to administer the database.
- Verify that the Oracle user (at the operating system level) owns all of the files in $ORACLE_HOME/bin. Check permissions in this directory and (on UNIX) check the umask value. File permissions should be 0750 or less.
- Understand what features and packages are installed on your system. Oracle is very functional and has many options. If you're installing from scratch, install only those features that you really need. If you already have an installation, review the options that are enabled and remove those that you don't need. The first principle of hardening (in

any environment) is that an option that is not installed cannot be used against you.

- Ensure limited file permissions for init.ora.
- Verify limited file permissions for webcache.xml, snmp_ro.ora, snmp_rw.ora, sqlnet.ora, htaccess, wdbsvr.app, and xsqlconfig.xml.
- Set HTTP passwords.
- Disable iSQL*Plus for production servers.
- Remove default accounts that are not used. (More on this in Chapter 4.)
  - There are many issues related to the SNMPAGENT user, so make sure this is one of the users that are removed (unless you really need to use it).
- Check for default passwords such as "`change_on_install.`" (More on this in Chapter 4.)
- Check that users are defined using strong passwords. This is especially important for SYS and for SYSTEM. (More on this in Chapter 4.)
- Use password profiles. (More on this in Chapter 4.)
- Close ports that are not needed. Don't use port redirection. Remove networking protocols that are not needed. (More on these in Chapter 3.)
- Ensure that the following values are set in init.ora:

```
_trace_files_public=FALSE

global_names=TRUE

remote_os_authent=FALSE

remote_os_roles=FALSE

remote_listener=""

sql92_security=TRUE
```

- On Windows, set the OSAUTH_PREFIX_DOMAIN registry key to true.
- Remove completely or limit privileges that include ANY.
- Limit or disallow privileges for ALTER SESSION, ALTER SYSTEM, and BECOME USER.
- Don't set default_tablespace or temporary_tablespace to SYSTEM for user accounts.
- Limit users who have a "DBA" granted role.

- Don't collapse OSDBA/SYSDBA, OSOPER/SYSOPER, and DBA into one role. Groups mapping to the OSDBA role, the OSOPER role, and the software owner should be distinct.
- Limit users who have "`with admin`" privileges. This will limit users who can change the schema and other system attributes.
- Limit "`with grant`" options. These create privilege chains in which a user is allowed to grant access to other users.
- Fully understand, monitor, and review system privileges assigned to users and roles. These are stored in `DBA_SYS_PRIVS`. Remember that you will get a list for both users and roles and that there is a hierarchical role structure. As an example, selecting `select * from dba_sys_privs where grantee='SYS'` will show all of the SYS system privileges:

```
GRANTEE    PRIVILEGE                            ADM
---------- ------------------------------------ ---
SYS        AUDIT ANY                            NO
SYS        DROP USER                            NO
SYS        RESUMABLE                            NO
SYS        ALTER USER                           NO
SYS        ANALYZE ANY                          NO
SYS        BECOME USER                          NO
SYS        CREATE ROLE                          NO
SYS        CREATE RULE                          YES
…
SYS        ADMINISTER DATABASE TRIGGER          NO
SYS        ADMINISTER RESOURCE MANAGER          NO
SYS        CREATE PUBLIC DATABASE LINK          NO
SYS        DROP ANY EVALUATION CONTEXT          YES
SYS        ALTER ANY EVALUATION CONTEXT         YES
SYS        CREATE ANY EVALUATION CONTEXT        YES
SYS        EXECUTE ANY EVALUATION CONTEXT       YES

139 rows selected.
```

- Make sure that the utl_file_dir parameter in V$PARAMETER is not set to * or to the same value as that for user_dump_dest.
- Limit as much as possible permission to the SGA tables and views. Users have no business accessing the X$ tables, DBA_ views, or V$ views, and there is too much sensitive information in these objects that would be a paradise for hackers.

- Limit as much as possible access to ALL_USERS and all the ALL_% views.
- Limit access to SYS.AUD$, SYS.USER_HISTORY$, SYS.LINK$, SYS_USER$, SYS.RESOURCE$, PERFSTAT.STAT$SQLTEXT, PERFSTAT.STATS$SQL_SUMMARY, ALL_SOURCE, DBA_ROLES, DBA_SYS_PRIVS, DBA_ROLE_PRIVS, DBA_TAB_PRIVS, DBA_USERS, ROLE_ROLE_PRIVS, USER_TAB_PRIVS, and USER_ROLE_PRIVS.
- Secure access to catalog roles and dba role views.
- Revoke public execute privileges on utl_file, utl_tcp, utl_http, utl_snmp, dbms_random, dbms_lob, dbms_job, dbms_scheduler, owa_util, dbms_sql, and dbms_sys_sql.
- Revoke CONNECT and RESOURCE roles from all users.
- Check all database links and make sure you are not storing passwords in clear text. (More on this in Chapter 8.)
- Set a password for the listener. (More on this in Chapter 3.)
- Remove the EXTPROC entry from listener.ora. (More on this in Chapter 7.)
- Use product profiles to secure SQL*Plus. (More on this in Chapter 5.)
- Set tcp.validnode_checking, tcp.invited_nodes, and tcp.excluded_nodes in protocol.ora (Oracle 8i) or sqlnet.ora (Oracle 9i,10g). (More on this in Chapter 5.)
- Revoke as many packages from PUBLIC as possible.
- Audit that developers cannot access production instances.
- Enable auditing. This is a complex topic. (More on this in Chapters 11 through 13.)

Once you have finished hardening your Oracle environment, you may want to validate your environment using the audit checklist available at www.petefinnigan.com/orasec.htm.

1.1.2 Hardening a SQL Server environment

SQL Server has suffered from a lot of bad press and from several very visible attacks. It is also one of the most functionally rich databases, which translates to "inherently insecure" in security lingo. Luckily, SQL Server is also one of the most well-documented environments. There are numerous resources available that can help you secure your SQL Server environments, many products that can be of assistance, and a very large community supporting security in this environment. Furthermore, contrary to public perception, Microsoft is actually investing a lot in making the SQL Server platform more secure.

Hardening a SQL Server environment should include at least the following tasks:

- Physically secure the server on which SQL Server lives.
- Apply all service packs and hot fixes to both the Windows operating system and SQL Server. You can execute `select @@version` to see precisely which version you are running. You can see what this version maps to in terms of patch levels at www.sqlsecurity.com/DesktopDefault.aspx?tabid=37.
- Make sure all SQL Server data files and system files are installed on an NTFS partition and that the appropriate permissions are defined for the files.
- Use a low-privilege user account for the SQL Server service. Don't use LocalSystem or Administrator.
- Delete setup files. Setup files may contain plain text and weakly encrypted credentials. They contain sensitive configuration information that has been logged during installation. These files include sqlstp.log, sqlsp.log, and setup.iss in the MSSQL\Install (or MSSQL$<instance name>\Install). Microsoft provides a free utility called killpwd that locates and removes these passwords from your system.
- Secure the sa account with a strong password.
- Remove all sample users and sample databases.
- Review all passwords. At the very least, check for null passwords using the following SQL: `select name, password from syslogins where password is null`. (See Chapter 4 for more on password strength.)
- Remove the guest user from all databases except from master and tempdb.
- Review how roles are assigned to users at a database and server level and limit assignment to the minimal set necessary.

- Put a process in place that allows you to periodically review role and group membership.
- Use Windows authentication rather than mixed authentication.
- Remove network libraries that are not used (or that you don't know are used). SQL Server can be accessed using several network libraries. Most environments are based on TCP/IP, in which case all other network libraries should be removed. (More on this in Chapter 3.)
- Require all access to the database server to be networked. Don't allow or promote remote access to the operating system and running tools locally.
- Remove or restrict access to extended (xp__ stored procedures). Restrictions can be to administrator accounts only or in some cases even more restrictive. (See Chapter 7 for more details.)
- Do not install user-created extended procedures because they run with full security rights on the server.
- Check and limit procedures that are available to PUBLIC. To check which procedures may be a problem, you can use the following SQL: `select sysobjects.name from sysobjects, sysprotects where sysprotects.uid = 0 and xtype IN ('X','P') and sysobjects.id = sysprotects.id`.
- Disable SQL mail capabilities and find alternative solutions to notification methods.
- Do not install full-text search unless your application requires it.
- Disable Microsoft Distributed Transaction Coordinator unless distributed transactions are really required for your application.
- Check for startup Trojans. Make sure there are no weird calls in `master..sp_helpstartup`. (See Chapter 9 for more details.)
- Check for password-related Trojans by comparing `master..sp_password` to that of a fresh install. (See Chapter 9 for more details.)
- Closely monitor all failed login attempts. Put together the procedure and process for giving you constant access to this information. (More on this in Chapters 4 and 12.)
- Audit that developers cannot access production instances.
- Enable auditing. This is a complex topic. (More on this later in this chapter and in Chapters 11 through 13.)

An excellent resource for hardening SQL Server is a script written by Chip Andrews that you can download from www.sqlsecurity.com/DesktopDefault.aspx?tabid=25 (or go to www.sqlsecurity.com and select Tools -> Lockdown Script from the menu bar).

1.1.3 Hardening a DB2 UDB (LUW) environment

- Physically secure the server on which the DB2 instance lives.
- Do not run DB2 as root (or as LocalSystem on Windows). On Windows, run the service as a local nonprivileged user and lock down registry permissions on DB2 keys.
- Verify that all DB2 files have restrictive file permissions. On UNIX this means 0750 or more restrictive.
- Remove default accounts that are not used.
- Remove the sample database and any other databases that are not needed.
- Check for default passwords. Check password strengths, especially in db2admin, db2inst?, db2fenc?, and db2as. (More on this in Chapter 4.)
- Enable password profiles (lockout and expiration).
- Never use CLIENT authentication. Use SERVER_ENCRYPT, DCE_ENCRYPT, or KRB_SERVER_ENCRYPT if possible. (More on this in Chapter 4.)
- Close unnecessary ports and services (e.g., the JDBC applet service and ports 6789 and 6790).
- Remove all permissions granted to PUBLIC. At the very least, revoke IMPLICIT_SCHEMA database authority from PUBLIC.
- Restrict who has SYSADM privileges. The installation may assign SYSADM privileges to too many of the default users, and it is important to remove these privileges.
- Revoke privileges on system catalogs: SYSCAT.COLAUTH, SYSCAT.DBAUTH, SYSCAT.INDEXAUTH, SYSCAT.PACKAGEAUTH, SYSCAT.PASSTHRUAUTH, SYSCAT.ROUTINEAUTH, SYSCAT.SCHEMAAUTH, and SYSCAT.TABAUTH.
- If running on Windows, add all normal users to the DB2USERS group and all administrators to the DB2ADMINS group.

- If running on Windows, change the user under which the DAS service runs using `db2admin setid<username> <password>`. Don't use the services utility, because some of the required access rights will not be set for the logon account.
- Audit that developers cannot access production instances.
- Enable auditing. This is a complex topic. (More on this later in this chapter and in Chapters 11 through 13.)

1.1.4 Hardening a Sybase environment

- Physically secure the server on which Sybase lives.
- Apply all Emergency Bug Fixes (EBFs), Electronic Software Deliveries (ESDs), and Interim Releases (IRs) to both the operating system and to Sybase. You can execute `select ++version` and download appropriate patches from the Sybase support Web site.
- Ensure that the directories in which Sybase is installed can be accessed only by the administrator user.
- Secure the sa account with a strong password.
- Remove all sample databases and review which databases are available on the server. You can use `exec sp_helpdb`.
- Remove all system accounts that are not used and review password strengths for those that are left. Pay special attention to the following login names, which may exist as part of installations of other Sybase servers:

Name	Description
dba	Created with Enterprise Portal Express Edition
entldbdbo	Created with database access control
entldbreader	Created with database access control
jagadmin	Created with Enterprise Portal Application Server
pkiuser	Created with Enterprise Portal
PlAdmin	Created with Enterprise Portal Application Server
PortalAdmin	Created with Enterprise Portal
pso	Created with Enterprise Portal
sybmail	Created when the Sybase mail service in installed (it should *not* be installed—see the next bullet)

- Don't use the Sybase mail capability.
- Review all passwords. (See Chapter 4 for more on password strength.)
- Make sure that passwords are set to expire by setting `exec sp_configure "password expiration interval", 60`. You can use any number except 0, which means that passwords never expire. The example above sets passwords to expire after 60 days. (More on this in Chapter 4.)
- Require strong passwords. For example, set `exec sp_configure "password expiration interval", 1` to ensure that each password has at least one digit and set `exec sp_configure "minimum password length", 8` to ensure that each password is at least eight characters long (or whatever your policy requires). (More on this in Chapter 4.)
- Remove the guest user from all databases except from master and tempdb.
- If you are running a Windows-based system, verify that the Sybase registry keys have the appropriate permissions.
- If running on a Windows system, prefer integrated authentication mode. You can check the authentication mode using `exec sp_loginconfig "login mode"`. Integrated is a value of 1.
- Ensure that the default login (used in integrated login mode when a user has no entry in the `syslogins` table) is mapped to a low-privilege account or, preferably, to null. You can view the mapping using `exec sp_loginconfig "default account"`.
- Protect the source code of stored procedures, views, triggers, and constraints. Ensure that the `syscomments` table is protected by testing that the value for `exec sp_configure "select on syscomments.text"` is 0. (More on this in Chapter 9.)
- Ensure that users cannot write stored procedures that modify system tables. You can test the value using `exec sp_configure "allow updates to systems tables"`.
- Make sure resource limits are enabled by testing the value using `exec sp_configure "allow resource limit"`. You can then set resource limits per user (stored in sysresourcelimit). This protects your server against denial-of-service attacks because a user who has been granted access to the system cannot bring the server to its knees by issuing commands that generate huge result sets and otherwise consume too many resources.

- Closely monitor all failed login attempts. There are numerous ways to do this. (More on this in Chapters 4 and 12.) If you want to log these failed attempts to the error logs, use `exec sp_configure "log audit logon failure"`.
- When running on a Windows server, remove the xp_cmdshell extended procedure by executing `exec sp_dropextendedproc xp_cmdshell`.
- Audit that developers cannot access production instances.
- Install the Sybase auditing feature and use the auditing tables in sybsecurity or use other audit mechanisms. (More on this later in this section and in Chapter 11 through 13.)

1.1.5 Hardening a MySQL environment

Of the database platforms mentioned in this chapter, MySQL is the only open-source database platform. Being open source has advantages and disadvantages when dealing with security and hardening. In the long term, the open-source community has shown that the sheer number of users and the open sharing of information guarantees high levels of quality and therefore fewer vulnerabilities and better security. In the short term, however, open source means that hackers have access to the source code and can easily figure out the weaknesses of the product and how to exploit them. Regarding MySQL, we are still in the early days, and security for MySQL is a concern. Moreover, the new features recently introduced in version 5.0 will lead to more security issues, and security management in version 5.0 promises to be a challenge. A good starting point for MySQL hardening should include at least the following:

- Physically secure the server on which MySQL lives.
- Use the following mysqld options:
  - --local-infile=0 to disable LOCAL in LOAD DATA statements
  - --safe-show-database to ensure that a SHOW DATABASES command only lists databases for which the user has some kind of privilege. If you wish to be even more restrictive, use the --skip-show-database option.
  - --safe-user-create ensuring that a user cannot create new users using GRANT unless the user has INSERT privileges into MYSQL.USER

  - --secure-auth disallowing authentication for accounts that have passwords from versions prior to 4.1
  - --skip-name-resolve
  - --skip-symbolic-links disallows the use of symbolic links to tables on UNIX

- *Do not* use the --skip-grant-tables mysqld option.
- *Do not* use the --enable-named-pipe option on Windows—use TCP network access rather than named pipes. (More on this in Chapter 3.)
- *Do not* grant the PROCESS, FILE, or SUPER privileges to nonadministrative users.
- When using MySQL as a back-end for a Web server, don't run MySQL on the same host as the Web server. This has been suggested in some texts so that MySQL can be configured to disallow remote connections. However, the risks of having the database on the same host as the Web server are greater than the benefit in disallowing networked connections. For example, many Web servers have known vulnerabilities that would allow a hacker to download files, including for example MyISAM or innodb files used by MySQL.
- Ensure a strong password for user root.
- Disallow the default full control of the database to local users and disallow the default permissions for remote user to connect to the database. `delete from user where user ='';`
- Don't use MySQL prior to version 4.1.x; there are too many serious vulnerabilities in the authentication protocol. Prefer a version later that 4.1.2 because these do not suffer from a buffer overflow vulnerability that allows authentication bypass.
- Limit privileges to the load_file function.
- Limit privileges to load data infile and select into <file>.
- Disallow developer access to production instances.
- Enable auditing. This is a complex topic. (More on this later in this chapter and in Chapters 11 through 13.)

1.1.6 Use configuration scanners or audit checklists

After you harden your database environment, you need to periodically check that your database is still locked down and that no new misconfigurations have been introduced. This involves a continuous effort that can sometimes be automated with a set of tools. Sometimes this best practice may already be implemented by the information security group. For example, if you are running SQL Server, your security group may already be using Microsoft's Baseline Security Analyzer in the context of checking configurations of Windows and servers such as IIS and SQL Server. In this case you may be able to piggyback on their activities and include a continuous check for the database.

The Microsoft Baseline Security Analyzer (MBSA) is a tool that allows you to scan one or more Windows systems for common security misconfigurations. MBSA will scan a Windows-based computer and check the operating system and other installed components, such as Internet Information Services (IIS) and SQL Server. The scan checks for security misconfigurations and whether these servers are up-to-date with respect to recommended security updates. MBSA scans for security issues in SQL Server 7.0 and SQL Server 2000 (including MSDE instances) and checks things like the type of authentication mode, sa account password status, and SQL service account memberships. Descriptions of each SQL Server check are shown in the security reports with instructions on fixing any of the issues found. MBSA will help you with:

- *Checking members of the sysadmin role*. This check determines the number of members of the sysadmin role (giving system admin rights to the instance) and displays the results in the security report.
- *Checking restrictions of cmdexec rights*. This check ensures that the cmdexec right is restricted to sysadmin only. All other accounts that have the cmdexec right are listed on the security report. Because the SQL Server Agent can automate administrative tasks by using scripted jobs that can perform a wide range of activities, including running T-SQL scripts, command-line applications, and Microsoft ActiveX scripts, their execution should be limited to privileged users.
- *Checking SQL Server local account passwords*. This check determines whether any local SQL Server accounts have simple passwords, such as a blank password. This check also notifies you of any accounts that have been disabled or are currently locked out. Password checks include checks for:
    - Password is blank
    - Password is the same as the user account name
    - Password is the same as the machine name
    - Password uses the word "password"
    - Password uses the word "sa"

    - Password uses the word "admin" or "administrator"

- *Checking that Windows authentication is being used*.
- *Checking whether SQL Server BUILTIN\Administrators is a member of the sysadmin role*. This check determines whether the built-in Administrators group is listed as a member of the Sysadmin role. Fixed server roles have a server-wide scope. They exist outside of the database. Each member of a fixed server role is able to add other logins to that same role. All members of the Windows BUILTIN\Administrators group (the local administrator's group) are members of the sysadmin role by default, which gives them full access to all of your databases.
- *Checking SQL Server directory access*. This check verifies that a set of SQL Server directories has limited access to SQL service accounts and local Administrators only. The tool scans the access control list (ACL) on each of these folders and enumerates the users contained in the ACL. If any other users (aside from the SQL service accounts and Administrators) have access to read or modify these folders, the tool marks this check as a vulnerability. The directories scanned are:
    - Program Files\Microsoft SQL Server\MSSQL$InstanceName\Binn
    - Program Files\Microsoft SQL Server\MSSQL$InstanceName\Data
    - Program Files\Microsoft SQL Server\MSSQL\Binn
    - Program Files\Microsoft SQL Server\MSSQL\Data
- *Checking whether the sa account password is exposed*. This check determines whether SQL Server 7.0 SP1, SP2, or SP3 sa account passwords are written in plain text to the setup.iss and sqlstp.log\sqlspX.log files in the %windir% and %windir%\%temp% directories (this may happen when mixed authentication is used). The splstp.log\sqlspX.log file is also checked on SQL 2000 to see if domain credentials are used in starting the SQL Server services.
- *Checking the SQL Server guest account*. This check determines whether the SQL Server guest account has access to databases other than master, tempdb, and msdb. All databases to which the account has access are listed in the security report.
- *Checking whether SQL Server is running on a domain controller*. It is recommended that you do not run SQL Server on a domain controller. Domain controllers contain sensitive data such as user account information. If you run a SQL Server database on a domain control-

ler, you increase the complexity involved in securing the server and preventing an attack.

- *Checking SQL Server registry key security.* This check ensures that the Everyone group is restricted to read permission for registry keys, including HKLM\Software\Microsoft\Microsoft SQL Server and HKLM\Software\Microsoft\MSSQLServer. If the Everyone group has more than read permission to these keys, it will be flagged in the security scan report as a vulnerability.
- *Checking SQL Server service accounts*. This check determines whether the SQL Server service accounts are members of the local or domain administrators group on the scanned computer, or whether any SQL Server service accounts are running under the LocalSystem context. The MSSQLServer and SQLServerAgent service accounts are checked on the scanned computer.

1.2 Patch your database

One of the expressions used by information security professionals is that you should patch, patch, and then patch some more. Although patch management is not synonymous with security and certainly does not guarantee security, it is one of the most important and fundamental techniques, without which security does not exist. Software bugs are often exploited for launching an attack, and if there is a bug in the security layer (e.g., the bugs in MySQL's authentication systems prior to version 4.1.x), then database security is certainly a challenge. Moreover, it is hard enough to combat threats that use problems you may not know about. At the very least, patches help you address threats that are launched against *known* problems.

Patching is difficult and unfortunately has an inherent time delay during which your system is exposed to an attack. Some of this time delay results from your own schedules for testing and applying patches to production environments. Some of this delay involves vendors who don't necessarily release the patches quickly enough. As an example, IBM DB2 UDB Version 7.2 had a buffer overflow vulnerability in the LOAD and INVOKE commands. These vulnerabilities were acknowledged by IBM on November 22, 2002. The fix was available starting September 17, 2003—10 months later! This is not unique to IBM—any complex software takes time to fix, test and release. Therefore, patching is not a silver bullet, but it is a bullet nevertheless.

1.2.1 Track security bulletins

Knowing where your database environment is vulnerable and what patches are available to remediate these security problems is one of the most useful things you can do. This does not necessarily mean that for every published alert you must go through a patching process (nor does it mean that the vendor releases a hotfix for every vulnerability). However, you should always be aware of security issues, and you need to know when vulnerabilities apply to your environment.

Several Web sites track security vulnerabilities, alerts, and advisories, including vulnerabilities for database environments. The various sites often mirror each other in terms of the content—when a security alert is posted on one it is normally available on the others as well. Major security vendors also post security alerts as a service to their customers (and to promote themselves). While each person has a preference, these sites are a good starting point:

- *www.cert.org:* Established in 1988, the CERT Coordination Center (CERT/CC) is a center of Internet security expertise, located at the Software Engineering Institute, a federally funded research and development center operated by Carnegie Mellon University.
- *cve.mitre.org:* The Common Vulnerabilities and Exposures (CVE) is a list of standardized names for vulnerabilities and other information security exposures. CVE aims to standardize the names for all publicly known vulnerabilities and security exposures and is based on a community effort. The content of CVE is a result of a collaborative effort of the CVE Editorial Board. The Editorial Board includes representatives from numerous security-related organizations, such as security tool vendors, academic institutions, and government as well as other prominent security experts. The MITRE Corporation maintains CVE and moderates Editorial Board discussions. CVE is not a database; it is a list. The goal of CVE is to make it easier to share data across separate vulnerability databases and security tools. You will therefore see that vendors often map their IDs for vulnerabilities to a CVE number. These numbers will have a format similar to CAN-2003-0058 or CVE-2001-0001—the first one being a candidate as opposed to an entry accepted and cataloged into CVE.
- *www.securityfocus.com/bid:* A vendor-neutral site that provides objective, timely, and comprehensive security information to all members

of the security community, from end users, security hobbyists, and network administrators to security consultants, IT Managers, CIOs, and CSOs.

- *www.securitytracker.com/search/search.html:* SecurityTracker is a service that helps you keep track of the latest security vulnerabilities. You can also submit a vulnerability to bugs@securitytracker.com.

In addition to organizations such as CERT and repositories such as CVE that classify security alerts of all types, each vendor has its own security resource page:

- *Oracle:* The Oracle Security Alerts Page is at www.oracle.com/technology/deploy/security/alerts.htm.
- *SQL Server:* The SQL Server Security Center is at www.microsoft.com/technet/security/prodtech/dbsql/default.mspx.
- *DB2:* The DB2 support page is at www-306.ibm.com/software/data/db2/udb/support/.
- *Sybase:* The Sybase support page is at www.sybase.com/support and the support ASE page is at www.sybase.com/products/information-management/adaptiveserverenterprise/technicalsupport.

You can subscribe to security alerts for each of the main database platforms:

- *Oracle:* www.oracle.com/technology/deploy/security/securityemail.html
- *SQL Server:* www.microsoft.com/technet/security/bulletin/notify.mspx
- *DB2:* Register for the My Support program at www-1.ibm.com/support/mysupport/us/en/.
- *Sybase:* Register for MySybase notifications from a link on the Sybase support page at www.sybase.com/support.

The user community for each of the major database platforms is quite large, and while learning that your product has a flaw and is vulnerable to an attack is certainly not fun, all vendors realize that if the community notifies them of the problem, they can fix it and better support their customers. If you find a vulnerability, you can report them to the following resources:

- *Oracle:* E-mail to SECALERT_US@ORACLE.COM
- *SQL Server:* https://s.microsoft.com/technet/security/bulletin/alertus.aspx
- *DB2:* www-306.ibm.com/software/support/probsub.html

Oracle even went out of its way back in 2001 and posted the following notice on many forums:

> **How to Contact Oracle with Security Vulnerabilities**
>
> Oracle sincerely regrets the difficulty that its user community—its customers, partners and all other interested parties—has recently had in notifying Oracle of security vulnerabilities in its products and locating subsequent patches for these vulnerabilities.
>
> Oracle has taken the following corrective measures to facilitate notification of security vulnerabilities and location of security patch information. Oracle will post Security Alerts on Oracle Technology Network at URL: otn.oracle.com/deploy/security/alerts.htm. (A Security Alert contains a brief description of the vulnerability, the risk associated with it, workarounds and patch availability.) This URL also provides mechanisms for supported customers to directly submit a perceived security vulnerability in the form of an iTAR (Technical Assistance Request) to Oracle Worldwide Support Services. Those individuals who are not supported customers but who wish to report a vulnerability can directly email Oracle at SECALERT_US@ORACLE.COM with the details of the security vulnerability.
>
> Oracle believes that these mechanisms make maximum use of its existing customer support services, yet allow non-supported Oracle users and security-interested parties to contact Oracle directly and swiftly with information about security vulnerabilities.
>
> Oracle proactively treats security issues with the highest priority and reiterates that it is committed to providing robust security in its products. Oracle wishes to thank its user community for its patience and understanding and appreciates cooperation in this collaborative endeavor.

1.2.2 Example of a class of vulnerabilities: Buffer overflows

Although many types of vulnerabilities and attacks can affect a database (or any server for that matter), the class of vulnerabilities called *buffer overflows* has earned a prominent role in the history of information security. It is perhaps the most well known and most illustrious type of attack there is, and buffer overflow problems have almost become synonymous with the term *security vulnerability*. If you do a query on the CERT Web site, you will find 48 buffer overflow vulnerability notes related to Oracle and 13 buffer overflow vulnerability notes related to SQL Server. If you look at the Oracle Security Alerts page (www.oracle.com/technology/deploy/security/alerts.htm), you will find that of the 60 alerts listed, 16 are buffer overflow alerts. DB2 UDB 7.2 had a buffer overflow vulnerability in the INVOKE command and in the LOAD command. Versions 6 and 7 of DB2 had a buffer overflow vulnerability in db2ckpw that may let local users gain root access on the system. Sybase ASE has buffer overflow vulnerabilities in DBCC CHECKVERIFY, in DROP DATABASE, and in XP_FREEDLL. If you look at the number of buffer overflow vulnerabilities in general, you will find more than 660 different vulnerability notes on the CERT Web site.

If you look deeper into what components of a database these problems exist in, you may be surprised to find that it is very widespread. As an example, looking through the Oracle buffer overflow vulnerability notes will show that these exist in the listener, in the Oracle process itself (e.g., VU#953746), in functions (e.g., VU#840666), in the mechanism used for calling external procedures (e.g., VU#936868), in command-line programs (e.g., VU#496340), and more.

Any complex software usually has buffer overflow vulnerabilities, and databases certainly are highly complex programs. This is a direct consequence of the fact that buffer overflow vulnerabilities exist when developers do not validate the length of data that is used to reference a buffer or when they don't validate data that is copied into a buffer. Because this type of validation is easy to overlook and because many development environments are not always security conscious (in terms of coding best practices), this problem is very widespread. Although it is not the purpose of this chapter to teach you these coding best practices, it is a good idea to understand what a buffer overflow vulnerability really is, because you will encounter this term frequently if you adopt the habit of looking at security alerts (and patching your environment).

1.2.3 Anatomy of buffer overflow vulnerabilities

Buffer overflows are most common in languages such as C or C++, where arrays and pointers are the bread and butter of programming (and certainly, all of the major databases are written in C/C++). The simplest buffer overflow problem occurs when you have code that looks like:

```
char buf[100];
…
buf[111] = 'a';
```

In this case an array of size 100 was created but then the 111th location was dereferenced and written over. Another simple example occurs in the following code:

```
char buf[10];
…
strcpy(buf, "abcdefghijklmnopqrstuvwxyz");
```

Both of these code fragments are perfectly correct from a syntactic perspective and will not cause any problems for C and C++ compilers. However, these programs have an undefined result from a C/C++ language perspective, meaning that they may work sometimes and usually will wreak havoc within the program. The reason is that this code oversteps memory that may belong to another variable or that may be used by other elements in the program.

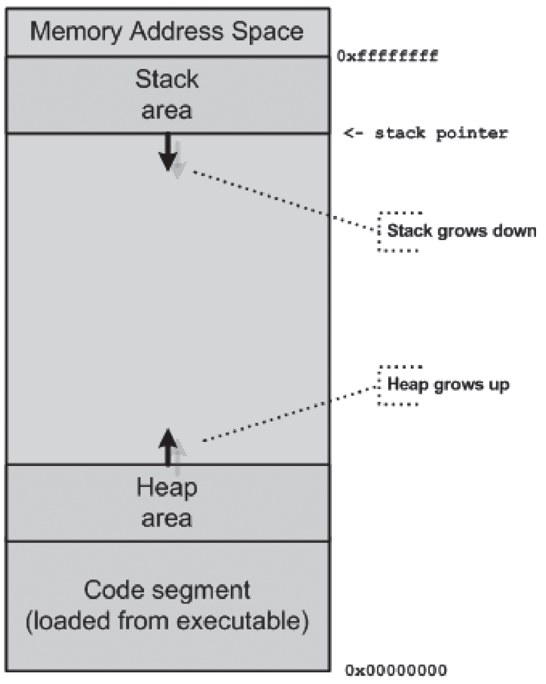
Before we move on to understand how this simple bug can be used by an attacker, it is worthwhile mentioning that the two code fragments shown previously are examples of problems that create stack buffer overflow vulnerabilities. There is a second class of buffer overflow problems that involve the heap and that occur when a developer would use `char *buf = malloc(10)` rather than `char buf[100]`, but in general stack-based buffer overflow vulnerabilities are more common and the principles are not very different.

In order to understand why overflows are such a big security problem, you need to remind yourself of how the operating system manages memory on behalf of a process. Any program needs memory to perform its tasks, and memory is usually divided into three main types:

1. Memory that is fixed for the program such as the code itself, static data, and read-only data
2. The heap, which is used when a program dynamically allocates memory using malloc or using new
3. The stack, which is used when calling methods and functions

In order to use all memory allotted for a process by the operating system, most computers manage the process memory as shown in Figure 1.1. The fixed parts (including the code and any static memory) are loaded at the bottom of the address space (usually not exactly at 0x00000000 but not far from it). Then comes the heap, which grows from low addresses to high addresses. If you continuously allocate variables on the heap, they will increasingly live in higher memory. Because both the heap and the stack may dynamically grow (the heap when you allocate more memory and the stack when you make more function calls), the operating system maximizes the usage of memory (and minimizes the work it has to do) by making the stack grow from high address spaces to low address spaces. As an example, if your main() calls foo(), which in turn calls bar(), and then your stack will include the segments for each of these functions, as shown in Figure 1.2.

Figure 1.1
Memory layout for an operating system process.



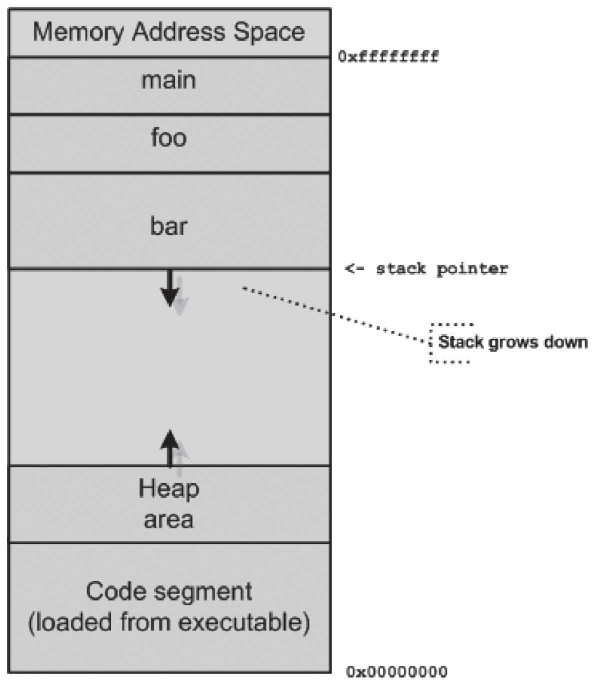


Figure 1.2 *Stack grows down (from high memory to low memory).*

The stack is used to manage the entire function calling process, including parameter passing and return values. When a function is called, the function's parameters are pushed onto the stack. Then an area is allocated for the return address. Then the frame pointer is pushed onto the stack (the frame pointer is used to reference the local variables and the function parameters that are always at fixed offsets from the frame pointer). Then the function's local automatic variables are allocated. At this point the program can run the function's code and has all of the required variables available. As an example, if you call a function foo("ab", "cd") that is defined as shown, the stack structure will include allocations, as shown in Figure 1.3.

```
int foo(char* a, char* b) {
 char buf[10];
 // now comes the code
 ...
}
```

Suppose that the first thing that the developer of foo does is copy the first parameter into the local variable so that he or she can manipulate the

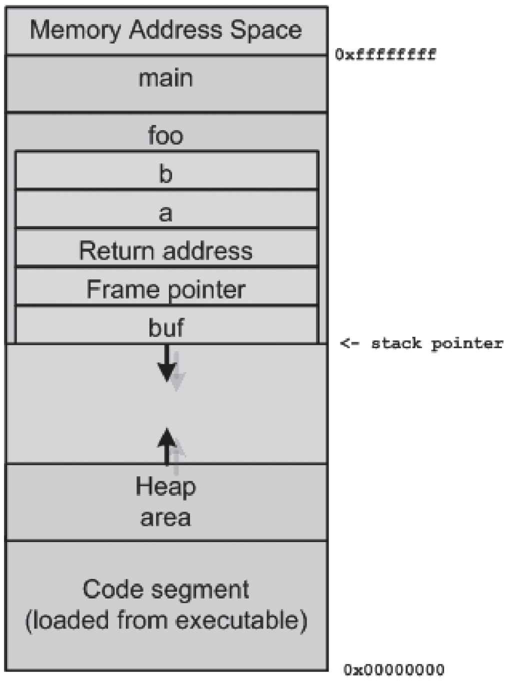


Figure 1.3 *Stack allocations when calling foo.*

data. Assume also that no-bounds checking is done and that the code looks like the following:

```
int foo(char* a, char* b) {
 char buf[10];
 // now comes the code
 strcpy(buf, a);
 ...
}
```

Foo has a buffer overflow vulnerability. In order to understand this, ask yourself what would happen if I were to call the function using:

```
main() {
 …
 int i = foo("I am a string that has many more characters than
10 and I will wreak havoc on your program", "ta da!");
 …
}
```

The result of this call is undefined. If you look at the memory layout, you will see that when the `strcpy` is performed, the long string starts out in the area allocated for buf, but because the stack grows top-down and the `strcpy` copies bottom-up, the string will start overwriting the frame pointer, then the return address area, and more. This will in many cases corrupt the stack and can easily cause your program to fail. Therefore, one type of attack that exploits buffer overflow vulnerabilities is a simple denial-of-service attack (vandalism). However, sophisticated hackers will use this vulnerability for something much more useful—for running their own code. Specifically, hackers will try to craft a string that, when overwriting the memory on the stack, will place malicious code and then overwrite the return address on the stack. When the function completes and the stack is unwound, the program will jump to the address of the malicious code (because the hacker has placed that return address there). This is not a simple thing to do, and the details are beyond the scope of this section. For an excellent paper that shows you how this can be done, refer to Aleph One's paper called "Smashing the Stack for Fun and Profit" (www.phrack.org/show.php?p=49&a=14).

Note that in a database environment the arbitrary malicious code is injected by the hacker into the program that has the buffer overflow vulnerability. In many cases this is the database server process, and the malicious code will have the same permissions as the database process.

1.3 Audit the database

There is no security without audit, and there is no need to audit without the need for security. For example, the term *C2 auditing* is often used independently, whereas it is really the auditing complement to a security classification called *C2 security* (see Appendix 1.A for a brief overview on C2 security). If you are serious about either one of these, you should implement both security and auditing in an integrated fashion.

Auditing plays both an active role and a passive role in security. By auditing database activity and access, you can identify security issues and resolve them quickly. The auditing function also serves to create checks and balances to ensure that an oversight does not cause the security layers to become invalid or ineffective. Finally, the fact that a database environment is being closely watched and audited causes a security layer based on deterrence—a very effective method in many environments.

On the flip side, auditing is not a goal but a means to elevate the security of your environment or to elevate the reliability and availability of your environment. In the context of this book, auditing is one of the most important security techniques. In fact, page-for-page, it is described in more detail than any other security technique covered in this book.

1.4 Define an access policy as the center of your database security and auditing initiative

Throughout this chapter you will learn about many domains with which you can start an implementation of database security and/or auditing. For example, you can start with network security and address protection of your database from remote attacks. You can start with a user-oriented approach and put provisions for increased security for privileged users such as DBAs. You can tackle issues that relate to the ways applications use your database and can even tackle the implementation layer by layer—starting with the authentication layer, moving to the authorization layer, and so on.

Regardless of how you choose to start, you should realize that database security is a complex topic, and there are many items to address. In order to ensure a successful implementation and avoid many frustrations, you should base the entire implementation on the concept of defining and implementing a security policy for your database environment. This will ensure that you do not lose sight of the big picture and the end goals, and

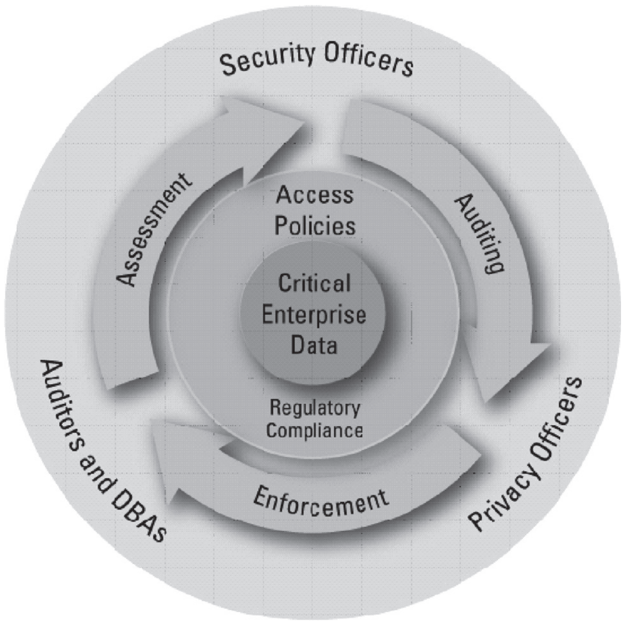


Figure 1.4
A database access policy is the core of any implementation.

that your investments in what are often disparate layers and techniques all work together toward the same goal. Additionally, any database security implementation will involve multiple people from multiple departments (e.g., DBAs, developers, information security officers, and auditors). A well-documented database usage security policy will also ensure that these individuals (who often have different skills and competencies) can use a common terminology and can augment each other rather than combat each other.

1.5 Resources and Further Reading

After you complete reading this book, here are additional resources (online resources and books) that can help you when implementing security and auditing initiatives that involve your database environments:

Oracle:

- *www.petefinnigan.com:* Pete Finnigan is one of the world's foremost Oracle security experts, and he posts a lot of useful information on his Web site.
  - *www.petefinnigan.com/weblog/archives:* Pete Finnigan's Oracle security weblog
- *www.dba-oracle.com/articles.htm#burleson_arts:* Many good articles on Oracle (and some on Oracle security) published by Don Burleson
- *www.linuxexposed.com:* A good resource for security including an excellent paper "Exploiting and Protecting Oracle" (http://files.linux-exposed.com/linuxexposed.com/files/oracle-security.pdf#search='pentest%20exploiting%20and%20protecting%20oracle')
- *www.appsecinc.com/techdocs/whitepapers.html:* Application Security Inc.'s white paper page, including a white paper titled "Protecting Oracle Databases"
- *www.dbasupport.com:* Miscellaneous articles, resources, and tips on Oracle
- *Oracle Security Handbook* by Marlene Theriault and Aaron Newman
- *Effective Oracle Database 10g Security by Design* by David Knox
- *Oracle Privacy Security Auditing* by Arup Nanda and Donald Burleson

SQL Server:

- *www.sqlsecurity.com:* Web site dedicated to SQL Server security
- *www.winnetmag.com/SQLServer/Security:* SQL Server Magazine's security page
- *http://vyaskn.tripod.com/sql_server_security_best_practices.htm:* Overview of SQL Server security model and best practices
- *www.appsecinc.com/techdocs/whitepapers.html:* Application Security Inc.'s white paper page, including a white paper titled "Hunting Flaws in Microsoft SQL Server White Paper"
- *SQL Server Security* by Chip Andrews, David Litchfield, Bill Grindlay, and Next Generation Security Software

DB2:

- *www.databasejournal.com/features/db2:* Database Journal for DB2
- *www.db2mag.com:* DB2 Magazine
- *www.appsecinc.com/techdocs/presentations.html:* Presentations on various topics, including "Hacker-proofing DB2"

Sybase:

- *www.isug.com/ISUG3/Index.html:* Sybase user group

MySQL:

- *www.nextgenss.com/papers.htm:* Papers on various topics, including MySQL security (e.g., "Hacker-proofing MySQL").
- *http://dev.mysql.com/doc/mysql/en/Security.html:* Security section from MySQL manual
- *www.appsecinc.com/techdocs/presentations.html:* Presentations on various topics, including "Hacker-proofing MySQL"

Hardening Linux:

- Hardening Linux by John Terpstra, et al
- Hardening Linux by James Turnbull

Hardening Windows:

- Hardening Windows Systems by Roberta Bragg
- Hardening Windows by Jonathan Hasell

Hardening Solaris:

- http://www.boran.com/security/sp/Solaris_hardening.html

Hardening AIX:

- A great IBM whitepaper is available at http://www-1.ibm.com/servers/aix/whitepapers/aix_security.html

Hardening HP/UX:

- www.securit.eclipse.co.uk/whitepapers/HPUX Hardening Guide.pdf
- www.hp.com/products1/unix/operating/security

1.6 Summary

In this chapter you learned some important first steps in securing your database environments. You learned how to harden your database environment and the importance of security alerts and of patching. You also got a glimpse into the world of database vulnerabilities and an example of how one class of vulnerabilities work. However, all of this is just an introduction.

In Chapter 2 you will continue in intro-mode and will get a glimpse into categories and domains of the security industry that are relevant to an effective implementation of database security and auditing. Chapter 3 is where the fun begins; this is when you will start to delve deeper into database security.

1.A C2 Security and C2 Auditing

C2 security is a government rating for security in which the system has been certified for *discretionary resource protection and auditing capabilities*. For example, SQL Server has a C2 certification, but this certification is only valid for a certain evaluated configuration. You *must* install SQL Server in accordance with the evaluated configuration or you cannot claim to be running a C2-level system. You can, however, be using C2 auditing in a system that is not C2-certified.

In order for a system to be certified with a C2 classification, it must be able to identify a user. Therefore, any C2-level system must implement the notion of user credentials (e.g., username and a password), must require a user to login using these credentials, must have a well-defined process by which a user enters these credentials, and must protect these credentials from capture by an attacker.

In a C2-certified system, users are accountable for their activities and any process they initiate. In order for this to be possible, any C2-certified system must be able to audit any user activity, including any attempt to read, write, and execute a resource managed by the system.

The next requirement of a C2-level system is that an owner of an object can grant permissions for access to the object for other users or groups. This is what the term *discretionary* implies. The default access for any object is no access other than the owner. If an administrator takes control over an object, the owner must know about this.

There are many other requirements for a system to be given a C2 certification, but many of them are not dealt with within the database security model but rather within the operating system's security model (e.g., protection for memory spaces, files, preemption of processing).

If you are running SQL Server, most chances are that you care more about C2 auditing than you do about C2 certification (unless you work for a government agency). C2 auditing tracks C2 audit events and records them to a file in the \mssql\data directory for default instances of SQL Server 2000, or the \mssql$instancename\data directory for named instances of SQL Server 2000. If the file reaches a size limit of 200 megabytes, C2 auditing will start a new file, close the old file, and write all new audit records to the new file.

To enable C2 auditing, you must be a member of the sysadmin role and you need to use the sp_configure system stored procedure to set `show advanced options` to 1. Then set `c2 audit mode` to 1 and restart the server. In a C2 certification, auditing is a must. Therefore, C2 auditing is implemented in a way that if auditing cannot occur, the entire database shuts down. For example, if the audit directory fills up, the instance of SQL Server will be stopped! You must be aware of this and take appropriate measures to avoid outage. Moreover, when you restart the instance of SQL Server, auditing is set to start up automatically, so you must remember to free up disk space for the audit log before you can restart the instance of SQL Server (or start the instance with the –f flag to bypass all auditing altogether). To stop C2 audit tracing, set `c2 audit mode` to 0. Finally, remember the following (extracted from SQL Server documentation):

Important: If all audit counters are turned on for all objects, there could be a significant performance impact on the server.

Computers / Data management / Security

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 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: DBA's, System and Network Administrators and Auditors

ISBN:1-55558-334-2
90000
9 781555 583347

ELSEVIER
DIGITAL
PRESS

books.elsevier.com/digitalpress

Compliments of:

IBM

For more information contact:

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