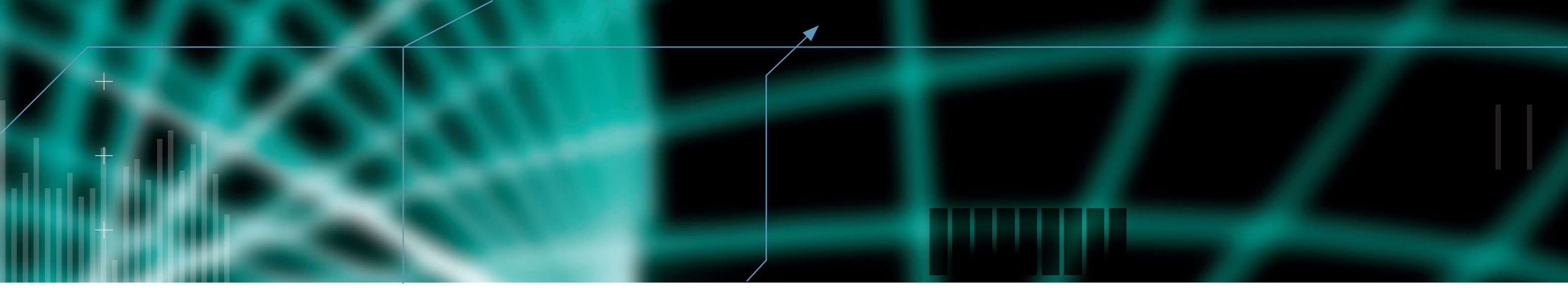

CPNI
Centre for the Protection
of National Infrastructure

NEXT GENERATION
SECURITY SOFTWARE

CPNI TECHNICAL NOTE

UNDERSTANDING DATABASE SECURITY

JULY 2008

KEY POINTS

- **Databases can be vulnerable to electronic attack**
- **The impact can be data theft, loss of service or data modification**
- **Mitigation measures include:**
- **Prevent direct access with a well configured firewall**
- **Encrypt both traffic and data**
- **Harden the operating system**
- **Remove all unused components**
- **Delete all unused accounts**
- **Change default passwords**
- **Design applications securely**
- **Use the principle of least privilege**
- **Perform regular security assessments**
- **Keep up to date with patches**

INTRODUCTION

This paper has been written to help database administrators, security professionals and risk framers understand the issues involved with database security and provide options and solutions when designing defensive strategies.

Security weaknesses in a computer system may impinge on one, two or all of that system's confidentiality, integrity or availability [1]. A loss of confidentiality to database servers implies unauthorized access; a loss of integrity implies unauthorized modification or deletion of data; and a loss of availability implies that database services cannot be accessed. Depending upon the purpose of the system involved in a given security incident, the loss of one of these may be considered to have a greater cost associated than another.

For example, the DBA for a database server that processes £1 million worth of transactions an hour will be more concerned with a loss of availability than say confidentiality as the data in this case is transient and uptime is more important. On the other hand, a police system that stores the details of informants, for example, is more concerned with a loss of confidentiality. Availability becomes irrelevant in the face of an attacker gaining access to this data and a security administrator would rather the system be disconnected than the attacker succeeding.

In some cases, and in addition to confidentiality, integrity and availability, the 'surety' of a system must be taken into consideration as well. The surety of a system can be defined as a degree of assurance in the trustworthiness of a system. For example, a web application which is known to be vulnerable to cross site scripting issues may be less trusted by a person and therefore not used. This could lead to the loss of custom especially in e-commerce based applications. The cross site scripting flaw in and of itself presents no harm to the system that exhibits the flaw and affects only the client but it certainly affects the business due to a loss of consumer confidence. A loss of surety is less of a problem when it comes to database servers but it must be kept in mind as database systems should not be looked at in isolation but as part of a unified system that provides a set of business services.

Each organisation will have their own risk profile and it is important to understand how the confidentiality, integrity, availability and surety of a system might be impacted in the event of a security incident. Differing weight is given to each in order to more effectively direct resources; especially time and money; when developing security policies and plans, assessing and mitigating risk, or designing and implementing risk management strategies.

Assessing exposure to and mitigating risk

An organisation decides what data it needs to collect and maintain in order to carry out the functions that support the business, and protecting that data is paramount; it is the lifeblood of the business. The very first step in securing access to data is with the drawing up of a security policy and plan. A security policy details what the rules are in order to maintain the health and security of a given system and set of processes, whilst a security plan defines how those rules are to be implemented and how they are to be enforced. Specifically, a database security policy facilitates giving access to those users that need it whilst keeping out those that do not; further it defines how access to data is granted and how that is managed.

At a minimum the database security policy should determine:

- who the principal players are and what their roles are
- who should be given access to the server
- what the password policy is
- who should have what system privileges
- who should be granted access to what objects
- who should have membership of what roles
- what accesses should be audited
- what to do when a security flaw is found.

Performing security assessments of database servers

There are two considerations for database security assessments. The first is an assessment of vulnerabilities and the second is an assessment of compliance to policy. Compliance to policy means ensuring that what is articulated in the security policy matches how the server is configured. For example, if the security policy dictates that no user is to have Database Administrator (DBA) privileges other than Alice and Bob then this is verified by checking which user accounts have DBA privileges. Another example, if the security policy states that only members of the Human Resources (HR) role are to have select access to the HR related tables then this is verified.

The line between a compliance problem and a security problem is not distinct. Compliance issues and security weaknesses that can increase a database server's intrinsic exposure to risk and are generally rooted in one of six categories:

- server configuration weakness
- object privilege weakness
- system privilege weakness
- user account settings weakness
- software vulnerability
- architectural flaw.

An "object" means something that exists within a database, such a table or a procedure.

If an issue is discovered during a security assessment and belongs to one of the first four categories then it can easily be addressed by a security administrator or DBA; whereas flaws found in the last two categories often require a software patch from the vendor to fix.

There are two more areas that need to be considered:

- The database's host operating system security, and
- The network's security

Operating system security

No matter how a database is configured as far as security is concerned, if the Operating System (OS) upon which it runs is insecure then so too is the database server. The reader is directed to other CPNI documentation that discuss operating system security and the vendors' documentation for a full discussion on the specifics of the OS in question and how to secure it, Below is a brief summary of general steps to improve OS security.

Where possible, database servers should be on dedicated systems; that is, the only service running should be the database server. This helps reduce risk by reducing the server's attack surface. In some cases, it may be necessary for remote administration to have other services running such as SSH (not telnet if possible as it is insecure) or the Server Service running.

If possible, the user account used to run the database server service should be low privileged as far as the OS is concerned. In some cases, certainly on Microsoft Windows, some database servers must run as Local Systems which means that if an attacker gains control of the database server software they can then gain full control over the OS. If a service account can be used on Windows systems, the service account should be configured for local access only and network logons denied. This will help prevent brute force attacks against this account; account lockout should not be enabled for the service account as doing so could cause a denial of service. which could affect the server's availability. On Unix-based systems the account should be configured to have no shell so that remote logons are not possible.

With regards to file ownership the following is recommended, where possible. The account used to run the software should only be able to execute the files that form the basis of the database server software. This service account should not be able to modify the binaries. Thus if the server is compromised an attacker cannot replace executables or shared libraries with versions containing Trojans. The service account should be able to read and write to the data files, transaction log files, audit log files, and any others that are required for the correct operation of the database server. The service account should not be the owner, however. No other account, other than the owner should be able to read or write to the data files, transaction log files etc; if they can, an attacker could gain access to these files via the operating system and steal or modify data.

Lastly, the server should be hardened according to the vendor's security documentation and vendor supplied security patches should be tested and applied as soon as possible.

Network security

A great deal of database server risk can be mitigated at the network layer and this section discusses those measures that can be taken to improve the database server's environment.

Firewalls

For a more secure environment, direct access to a database server should be tightly controlled with the use of a well configured firewall. Where possible, all user queries should be channelled through an application server where the code and business logic is contained; ideally no user should be able to run arbitrary queries against the database server or directly access the database server's TCP port. This helps reduce risk by reducing attack surface and confining users to predefined application queries. If, however, users must be allowed to run arbitrary queries against the database server then one could consider using an application proxy firewall that understands SQL. Please see the section on database intrusion detection and prevention as use of an application proxy firewall may, in certain cases, increase risk.

Default TCP port

A security through obscurity measure - changing the default port that the server listens on, for example - will help protect the server from opportunistic attacks and worms. The table lists the default ports for common database servers:

RDBMS	TCP Port
Oracle	1521
Oracle Apps	1526
Microsoft SQL Server	1433
Microsoft SQL Server ("hidden")	2433
MySQL	3306
DB2	50000
Informix	1526
Postgresql	5432
Sybase	4100

Network encryption

In a non-switched environment, an attacker with access to the LAN and a network sniffer may be able to gain access to database credentials and sensitive data as it travels across the wire. To prevent such attacks database network traffic should be encrypted. Most database vendors provide a network encryption module such as Oracle's Advanced Security option or the encryption CSM (Communications Support Module) in Informix that can be used to secure data as it travels over the network; alternatively a technology such as IPSec, OpenSSL or SSH tunnelling can be used.

Database security

As already indicated, database security issues are rooted in one of six categories, namely server configuration, object privileges, system privileges, user account settings, software vulnerabilities and architectural issues.

Server configuration

Most database servers have a number of start-up parameters and how these are set can affect aspects of the security of a database server. Covering each parameter of each vendor's database server software is beyond the scope of this paper and the reader is directed to the vendor's documentation for more information, but by way of example considers the Oracle RDBMS. A number of start-up parameters affect the security of the Oracle RDBMS. The REMOTE_OS_AUTHENT parameter, for example, defines whether a user can log in using an operating system account. If this parameter is set to TRUE then it lets a user perform an OS authorised logon and this can create a security vulnerability.

Object privilege weaknesses

Object privileges define who can do what to what object. Objects include tables, views, functions, procedures, indexes, triggers, etc. Object privileges changed depending upon the object type. For example the object privileges for tables include SELECT, INSERT, DELETE and UPDATE, and for procedures and functions object privileges include EXECUTE and DEBUG. Object privileges are typically given and removed with the GRANT and REVOKE statements respectively. One of the best ways to reduce risk for a given database server is to review and revoke many of the default shipped object privileges. This helps to reduce attack surface for a given user or role.

A good example of this is in Oracle. For four years the privileged SYS owned DBMS_EXPORT_EXTENSION package was vulnerable to multiple SQL injection flaws. This package is executable by PUBLIC by default. Over those four years Oracle produced three patches for this package but each time they failed to fix it properly. If, instead of just installing the patch when the news first broke about the vulnerabilities, a DBA just revoked the EXECUTE permission from PUBLIC then they would not have been vulnerable to exploitation.

System privilege weakness

System privileges define who can do what in the database server, for example, create, alter or drop objects, stop or start the database, enable or disable triggers. Depending upon the database server in question, any one of a large number of system privileges can provide an attacker with the stepping stone to indirectly gain DBA or system administrator privileges. Examples of such attacks are provided [2]. Because system privileges can be abused to gain full control careful consideration should be given to who is granted such privileges.

User account settings weakness

Weaknesses with user account settings can trivially lead to a complete database compromise. Some database servers in the past shipped with high privileged accounts with no password or a default password that was well known to the hacking community.

For example, SQL Server has an account called “sa” and this had no password by default. Indeed, the SQL Spida worm took advantage of this fact and used it as its entry point into a system as it spread. Older versions of Oracle prior to 10gR2 installed with a SYS account with a password of CHANGE_ON_INSTALL and a SYSTEM account with a password of MANAGER. If the passwords for default accounts are not changed then an attacker can easily gain access to the system. Even low privileged default accounts provide an ‘in’ from where other security flaws can be exploited to gain full control. As well as changing the passwords, where possible, if the account is not used it should be dropped from the system.

If, however, the account hosts an application but doesn’t require an active log on then the account should be locked and the password changed. Indeed it is commercial good practice that the security policy for a database server should ensure that all passwords have a minimum length of at least eight characters and that each password should have high complexity requiring numbers, letters and if possible other text characters. If the database supports case sensitive passwords, for example, Oracle 11g, then this should be enabled.   Passwords should be changed regularly, though this may present some difficulty when it comes to application accounts also, password reuse should be disallowed.

Lastly, account lock out should be enabled for non-application accounts after a set number of failed logon attempts. If an attacker can gain network access to the database server then following these recommendations will help prevent brute force attacks against accounts from succeeding. Another user related weakness is role membership that is too permissive. Membership of the DBA or system administration roles should be kept to a minimum and given only to those user accounts that require this to do their job. Database servers that support role passwords requiring the user to supply the password before the role privileges can be acquired should be used.

Software vulnerabilities

Software vulnerabilities come in many different forms and classes and whilst some affect all types of software some are database specific. The next few sections cover some of the more common issues that typically affect database server software and applications.

Buffer overflow and format string vulnerabilities

A buffer overflow vulnerability occurs when a program attempts to copy too much data into a memory buffer. This causes the buffer to overflow resulting in excess data overwriting other data in memory. If the data which is overwritten controls the path of the program's execution then an attacker can exploit this to gain control over the server. Even if the overwritten data doesn't control the program's execution then an attacker may still be able to exploit the overflow by changing the data in such a way that the program's logic is warped or an unknown value is set to a known value – for example the unknown password hash for a high privileged database user is changed to a known hash allowing the attacker to then logon as that user.

There are many different types of overflow which can be defined by looking at where they occur. For example, stack based buffer overflows occur in the stack memory. These are trivial for an attacker to exploit because the stack contains program control data. Another common type of buffer overflow is a heap overflow, named so because they occur on a heap. Heap memory is dynamically allocated by the program depending on memory needs. Whilst heap overflows are more difficult to exploit they can still allow an attacker to gain full control over the vulnerable process.

All of the common database servers have suffered from buffer overflow vulnerabilities in the past and many may still have buffer overflow vulnerabilities yet to be discovered. The greatest risk is posed by buffer overflow vulnerabilities that can be exploited prior to authentication. For example, both Oracle and Informix suffered from an overly long username buffer overflow. Both could be exploited to gain full control over the server. Microsoft SQL Server 2000 suffered from a flaw in the name resolution service and was exploited by the Slammer worm in the January of 2003.

Besides installing patches, a well configured firewall and limiting access to trusted systems helps mitigate the risk of exploitation of pre-authentication overflows. If the buffer overflow exists in a SQL function or procedure, then a firewall will do little to mitigate the risk. Here, reduction of attack surface and the appropriate limiting of object privileges is the best defence.

There are some cases where it is not possible to prevent public from executing a function by revoking execute privileges such as the pwdencrypt buffer overflow in SQL Server 2000. In this case the only way to prevent exploitation is with a patch. On Unix-based systems there is a risk of local (i.e. a user that is logged on to the local system either at the console or through telnet or SSH) exploitation of buffer overflows in executables that run with a different user identifier (called “setuid executables”).

For example, DB2 suffered from a local issue where an over-long DB2LPORT environment variable would cause an overflow in the libdb2.so library which could be exploited through one of the setuid root binaries such as db2cacpy. Exploitation of this flaw allowed local attackers to trivially gain root privileges. Again, if a patch can not be installed then the next best line of defence is with file permissions. Removing all file permissions from ‘others’ will prevent exploitation. This can be done using the Unix “chmod” command.

A format string vulnerability occurs when user input is used as what is known as the format string for one of the C runtime printf() family of functions and FormatMessage() in the Windows 32 bit Application Programming Interface (Win32 API). Under normal operation this format string contains format specifiers that determine what the function’s output should look like – i.e. its format.

There is a special specifier however that, when used, writes the number of bytes that have been outputted by the function to a memory location of the programmer’s choosing. If this format string is supplied by a user, rather than the programmer, they can send this special specifier and thus write a number of their choosing to a location of their choosing. They do this by getting the function to output a specific number of bytes (the number they want to write) and by loading the format string with memory addresses (the location they want to write to). In doing so, the attacker can gain control over the program by overwriting program control data – i.e. the data that tells the program where to go or what to do next. This might be a function pointer or an exception handler for example.

Like buffer overflows, format string vulnerabilities are very common and all database servers have suffered from one or two in the past. Patching is the best way to protect against format string vulnerabilities but the risk can be mitigated in other ways, for example, limiting of object privileges and reduction of attack surface. For more information on format string vulnerabilities please see the “resources” section.

Session object/handle ‘Snarfing’ vulnerabilities

During the execution of stored procedures an object or a handle may be created to track resources and this handle may be accessible outside of the procedure that created it and persists until the session ends or the handle is closed. An example of these may be file handles, TCP ports or SQL cursors (i.e. handles to a private working area of memory). If such handles are created at a higher privilege level than the current session’s privileges and the user can interact with them, then it may be possible for an attacker to exploit this (i.e. ‘snarf’ the handle) to gain unauthorised access to data or in the worst case fully compromise the server. An example of such an attack is called dangling cursor snarfing.

In Oracle [3] this is where a high privileged procedure opens a cursor for use with the DBMS_SQL package and, in the event of an error causing an exception, fails to close the cursor leaving the cursor dangling in the session. An attacker can then ‘snarf’ or take control of this cursor and recycle it to their own ends and, because the cursor was created with higher privileges than the attacker, they can use it to gain access to data they may not have had access to. If such a flaw exists in the code shipped by the database vendor then a patch is required to fix the flaw. However, developers of applications can ensure their own code is not vulnerable to such issues by ensuring that all objects and handles are properly closed in the event of an exception; indeed they should be properly closed whether there is an exception or not. Microsoft SQL Server applications can also be vulnerable to cursor snarfing.

Database SQL injection vulnerabilities

SQL injection flaws are not solely the preserve of web applications. Database procedures and functions can also be vulnerable to SQL injection attacks. Indeed, of all the software vulnerabilities patched by Oracle over the past few years the vast majority have been PL/SQL injection flaws.

The problem of SQL injection stems from unsanitised input from the user being executed by a procedure or function owned or defined by another user that has more privileges than the user. By carefully crafting their input to the vulnerable procedure an attacker can trick it into taking actions that the developer never intended thus leading to the security flaw. SQL injection flaws can allow attackers to escalate their privileges, gain unauthorised access to data, or in certain cases running operating system commands, for example the sp_mscopyscriptfile procedure in Microsoft's SQL Server 2000.

The impact of SQL injection flaws can be mitigated in several ways. Firstly, keeping up to date with patches reduces risk. Secondly, the reduction of attack surface can go a long way to removing risk entirely. This is achieved by deleting unused components and applying the principle of least privilege. For example, in Oracle a well-known PL/SQL injection flaw existed in a procedure called VALIDATE_STMT in the DRILOAD package owned by the CTXSYS user that is part of the ConText application. By default the special group PUBLIC (i.e. everyone) could execute this package and attackers could exploit this flaw to gain DBA privileges. Those that did not require the ConText application and had deleted it would not be vulnerable to this flaw. Those that did require the ConText application but had tightened the object privileges by removing PUBLIC execute access then their risk would have been greatly reduced.

Thirdly, as well as reducing attack surface, the impact of the exploitation of SQL injection flaws can be further reduced by database triggers which can be used to restrict Data Definition Language (DDL) operations (used to create and manage database objects). Finally if the source code for procedures and functions is available then a code review should reveal any potential weaknesses which can then be fixed.

Protecting against SQL injection in web applications

Web-based SQL injection attacks are where an attacker can inject (piggyback) their own additional SQL into a predefined query made by a web application by embedding SQL tokens or elements in the input. Such flaws arise due to lack of input validation.

Validation is the first port of call for protecting against SQL injection data. All input to the application should be sanitised (query string parameters, POST data, cookies, hostnames, etc) and verified against the application's specifications. For example, if the application is expecting a number in one particular input point then that input should be checked to ensure that the input is indeed a number. If the number should be between a range of values then this should be checked – by sending too large a number an attacker can cause application failures or errors. If the input is expected to be string based, such as a username, then close attention should be paid to single quotes. They should either be doubled up – i.e. for every single quote in the string a second should be added; alternatively they could be stripped removing them from the query.

Another option is to replace them with another character such as a caret. Length limits on string input should be enforced; if a column in a table used in a query is only 50 characters long then the input should be limited to only 50 characters in length.

Where possible application queries should use parameterised queries also known as host/bind variables. Here, user input is attached to the query only after the database server has compiled the query tree and, as such, has no way of influencing the query and thus stops SQL injection attacks cold. The only time using bind variables/ parameterised queries may not be possible is when a table or column name is not known in advance, for example when the user's input forms part of a table name.

Using database stored procedures is often cited as another method of protecting against SQL injection flaws. This is incorrect. If the procedure is executed dynamically and the database server is capable of batching queries, such as Microsoft SQL Server then using stored procedures offers no protection. Furthermore, on all databases the procedures themselves can be vulnerable to SQL injection.

Database Trigger Abuse

Database triggers can be used to restrict activities, enforce policy or ensure adherence, for example by making sure data being inserted into a column matches a specified format. On some database servers, triggers execute with the privileges of the owner and, as such, if there are any flaws in the trigger then an attacker could, by exploiting the flaw, take actions as the owner. This can lead to an attacker gaining extra privileges. Whilst real world examples of trigger flaws are few and far between, there are six known examples in Oracle, namely SDO_DROP_USER_BEFORE, SDO_CMT_CBK_TRIG, SDO_LRS_TRIG_INS and SDO_GEOM_TRIG_INS1 all owned by MDSYS, CDC_DROP_CTABLE_BEFORE owned by SYS and RLMGR_TRUNCATE_MAINT owned by EXFSYS. Attacks against triggers can be mitigated by disabling them where possible, or by reviewing and fixing the code.

Logic errors

One of the more difficult to spot security weaknesses gaining more attention due to the falling numbers of buffer overflow vulnerabilities is the class of logic errors. One such example of a logic flaw is an issue in the MySQL authentication process in versions 4.1.0 to 4.1.2 and early versions of 5. If a client flag was set in the authentication packet then an attacker could send a zero length password, and because the function used to check the password would return a success in that case due to a logic error, the user was authenticated [4]. A similar problem affected MySQL prior to 3.23.11 where an attacker only needed to send one character of their password meaning any account could be brute forced within 32 guesses.

Library loading

Most database systems allow a user to extend the database's functionality by loading a shared object or dynamic link library (DLL) that exports the code for the extended functionality. The way in which some databases do it however lead to a vulnerability whereby a low privileged attacker can remotely load a Trojan DLL that executes code as the user account used to run the database server process.

For example, Postgresql 7.4 and earlier allowed a user to load an arbitrary library using the LOAD extension. By placing attack code in the DllMain or _init() functions on Windows and Linux respectively an attacker could gain control over the server. The same attack could be launched against Informix using the ifx_replace_module procedure and the ifx_load_internal function [6].

Older versions of Oracle have a slightly different vulnerability and will be discussed later. Library loading attacks are usually only possible because the default system and object privileges are too lax. By limiting these privileges a DBA can reduce the risk. Whilst in some cases an attacker might be able to upload a DLL as a Binary Large Object (BLOB) to a table from where they then export it to the database server's disk and then load it into memory, other times the attacker can only load the DLL remotely, for example by using a Uniform Naming Convention (UNC) path on a Windows system.

Good egress filtering preventing outbound connections from the firewall will help mitigate the risk of this flaw. DBAs of Windows systems should block outbound connections to TCP 139 and 445 and if the server has the Web-based Distributed Authoring and Versioning (WebDAV) redirector installed then, in the event the server can't access port 139 and 445, it will failover to TCP port 80 - the Web. Thus outbound to TCP port 80 should be blocked as well on the firewall.

Inference

One of the more subtle security flaws related to database servers, especially those that hold highly sensitive data, is inference. This is where someone can infer some aspect about data without necessarily being able to see the data. For example, consider a table that stores orders for army personnel and each order has a sequential ID, a timestamp and a classification level – secret and non-secret. A low privileged user who can only see non-secret orders can infer that if there is a "missing" ID number then a secret order has been issued:

```
SQL> SELECT OID, ODATE, OTEXT FROM ARMY_ORDERS;

OID  ODATE               OTEXT
..
33   10:00:00 14/02/2008     Polish Boots
34   10:01:30 14/02/2008     Clean Mess
36   10:02:30 14/02/2008     Get lunch from NAAFI
..
```

Here we can see that there is no order ID of 35 – from this we can infer that at some time between 10:01:30 and 10:02:30 on the February 14, 2008 a secret order was issued. In this example, the weakness is caused by the sequential order ID number. Other sources which facilitate inference include but are not limited to, violation of database constraints, trigger errors and rowid functions. When designing database applications where inference could be a problem, time should be spent thinking about such issues and mitigation strategies devised.

Architectural flaws

Architectural flaws typically manifest themselves in the way disparate components of the same software package communicate. Two examples of architectural flaws can be found in Microsoft's SQL

Server 7 and 2000 and Oracle 9iR2 and earlier. The SQL Server flaw lies in the SQL Agent which is used to restart the server in the event of a failure and run scheduled jobs. Because all users, as the 'public' role, could schedule a job they could get the Agent to connect to the database with higher privileges and execute the job. By removing the execute permissions from public on the sp_add_job and sp_add_job_step procedures the risk of this flaw being exploited would be reduced. This demonstrates a good example of how reducing object privileges can protect the database server.

In Oracle, external procedures as libraries can be loaded by a user. Under the covers this works by the database connecting to the Oracle Transparent Network Substrate (TNS) Listener component and requesting that the library be loaded. The TNS Listener launches another process called extproc, which loads the library. An attacker can connect to the TNS Listener and pretend to be the Oracle database server and cause the extproc process to load a library of their choosing. A real world attack would load msvcrt.dll or libc and call the system() function which takes an arbitrary command line and executes it. Thus an attacker without a user ID and password could gain full control over the server. Limiting connections to the database to only trusted hosts with the use of a firewall would mitigate the risk of this vulnerability.

Direct file access

One of the easiest ways of bypassing database enforced access control to data is by directly accessing the data file in which the data resides. Such an attack can be launched from within the database server itself using built in file functions or if the database contains a Java virtual machine, such as DB2, Informix and Oracle or the CLR in Microsoft SQL Server 2005, this can be used, too. Direct data access can be defeated by requiring strong encryption of data and protection of the decryption keys. One problem that can not be solved with encryption directly is direct access to the transaction logs however unless data is encrypted by the client before new data is inserted or updated. Of course, an attack via the operating system is also possible if access to database files is not configured correctly.

Persistence of deleted and updated data

When a row of data is deleted most database servers do not remove or zero the data; they simply mark the row as deleted. This means that data that may have been deleted for security reasons may still be accessible. The same is true of updated data; depending upon the data type being updated a new row may be created with the new data leaving the old data still present but just marked as 'free'. The same is true for database indices. Whilst this feature can be beneficial when it comes to investigating security breaches it may pose problems when it comes to privacy issues or sensitive or classified data.

Resilience: backup, recovery and incident response

As already discussed, the availability of a system affects its security and in the event of any failure, whether due to the result of an electronic attack or even a power or hardware failure, loss to the business can occur.

Improving resilience

Improving resilience to failure in the event of an incident is by far the most cost effective strategy in the event a failure occurs. Resilience can be improved easily by using Redundant Array of Inexpensive Disks levels 1 and 0 (RAID 10) when installing a database server. This means that mirroring and striping is employed so that, in the event of a single disk failing, access to all the data is still possible. Ensuring that an Uninterruptable Power Supply (UPS) is used, helps prevent unclean shutdowns in the event of a power failure which can lead to a loss of data especially in the case of uncommitted transactions. UPS can give DBAs time to shut down any servers cleanly in the event of a major power failure and the RAID can help with hardware failure. Most RDBMS software allows multiple database servers to be configured to act as one system in a cluster. Cost permitting, this drastically improves resilience because the failure of one node in the cluster should not affect the smooth operation of the other systems and uptime is maintained.

Disaster recovery planning

Knowing what to do in the event of an incident helps enormously when it comes to recovering from the incident and, as such, a set of well planned, disaster recovery procedures should be developed, tested and rehearsed. Not only are the procedures important but also the people involved; knowing who does what during a failure or an incident is perhaps more important. A designated security co-ordinator should be in charge of this and where possible, this security co-ordinator should be a board level director so executive decisions can be taken quickly.

Database backups

Having a backup of the database server's data and metadata is essential if expeditious recovery after a failure is to happen. There are various strategies available to DBAs when it comes to backups; and which is best depends upon the value of the data being backed up and the cost incurred due to its loss. The amount of data requiring back up will also influence an organisation's backup plans – backing up 40 terabytes of data poses more problems than backing up 40 gigabytes, for example. A general approach may be to perform full backups once a month and incremental/differential backups each evening during 'quiet' periods. Backups should be encrypted because, if an attacker can gain access to the physical backup discs or tapes, then they'll be able to gain access to the data.

Incident response planning

Incident response planning allows an organisation to recover from a security incident quickly but it also allows them to take measured, controlled actions that will preserve evidence. Untainted evidence is crucial if a prosecution of the criminal involved is to take place. The incident response plan lays out what needs to be done during an incident and which parties should be involved. For example, if an attack was underway and the attacker had gained unauthorised access to the database server what should be done? Should the plug be pulled or the attacker's session disconnected? All of these questions and more need to be answered in an incident response plan. Devising a plan is beyond the scope of this paper, however, the reader is directed to the resource section which lists some good sources of information.

Forensics: auditing, intrusion detection

The activities of database users should be monitored and audited and most database servers provide the facilities to do so. Audit records can usually be written to a database table or an operating system file, though neither of these options provides full safety in the event of a database compromise. If, for example, an attacker breaks in and manages to gain DBA or system administrator level access then they will be able to delete any trace of their activities from the audit record is a database table is used. As most database servers offer access to the file system if the user has the right privileges, they can remove any traces from the audit record if an operating system file is used instead. As such, where possible, the audit record should be written to a different computer on the network using a one-way system so that records can not subsequently be deleted. Generally this requires the use of a third party add-on to the RDBMS software to enable remote logging of audit information.

Where possible all database activities should be logged, but as this can consume a considerable amount of disk space, especially in busy systems, this may not be practical or affordable. At a minimum access to data that has been deemed as critical or sensitive as per the organisation’s database security policy, such as financial data, should be logged. In addition to any changes to the database server’s metadata. Typically this would require auditing DDL operations such as calls to CREATE, ALTER, DROP, GRANT and REVOKE. Be aware though that using a direct INSERT, UPDATE or DELETE an attacker with the right level of privilege can alter that database’s metadata without having to use a DDL statement: but as most database servers require DBA or system administrator access to update the server’s metadata tables directly, all DBA actions should be logged as well. Lastly, log on attempts, whether successful or not, should be logged.

The logging of audit information alone is not sufficient,. The audit log should be regularly reviewed. Special attention should be paid to failed attempts at data access or logons as this could indicate an attack.

Database intrusion detection and prevention

As far as network-based intrusion detection and intrusion prevention is concerned it can be very difficult to spot an attack due to the programmability of database servers. An attacker has a potentially infinite number of ways in which they can encode their attack for example by using T-SQL in Microsoft SQL Server or PL/SQL in Oracle. Furthermore, regardless of programmability, an attacker can hide their attack using any of the databases string functions or maths functions An attacker can also “upload” their attack code into a database table over time using INSERTs or UPDATEs and then by SELECTing them and incorporating into an attack at a later date. For example, consider the following SQL:

```
INSERT INTO TBL1 (A1, A2) VALUES (‘ZXhlYyBtYXN0ZXIuLnhwX2NtZH’,1337);
INSERT INTO TBL2 (B1, B2) VALUES (‘NoZWxsICduZXQgdXNlciBkYXZp’,1337);
INSERT INTO TBL3 (C1, C2) VALUES (‘ZCBwYXNzd29yZCAvYWRkJw==’,1337);

EXECUTE (BASE64_DECODE(
SELECT A1 FROM TBL1 WHERE A2=1337 ||
SELECT B1 FROM TBL2 WHERE B2=1337 ||
SELECT B1 FROM TBL3 WHERE C2=1337 ))
```

Here an attacker has Base64 encoded their attack string an INSERTed it into three different columns in three different tables. Later they SELECT the attack string, Base64 decodes it then executes it. The attack here executes the XP_CMDSHELL extended stored procedure to add a user to the system yet an intrusion detection or prevention system would have failed to notice this attack. Depending upon where a host based IDS/IPS solution sits in the 'stack' they too may have great difficulty in spotting attacks.

The real value of Intrusion Detection/Prevention Systems (IDS/IPS) is demonstrated when it comes to the investigation of database security breaches. Provided all connections and SQL queries are logged then a skilled investigator should be able to reconstruct what has happened from the IDS/IPS logs.

CONCLUSION

Understanding database security is paramount for all organisations. As the world becomes more connected our exposure to threat increases and one of the most valuable targets for criminals and foreign intelligence services is data. This paper has covered most of the issues that affect database security, and is intended only as a primer and is by no means comprehensive. The reader is invited to review the "Resource" section for more information. In summary, a Top Ten list for improved database security is as follows:

- Prevent direct access with a well configured firewall
- Encrypt both traffic and data
- Harden the OS
- Remove all unused components
- Delete all unused accounts
- Change Default Passwords
- Design application securely
- Use the principle of least privilege
- Perform regular security assessments
- Keep up to date with patches

References

[1] csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf
[2] www.databasesecurity.com/dbsec/ohh-indirect-privilege-escalation.pdf
[3] www.databasesecurity.com/dbsec/cursor-snarfing.pdf
[4] www.ngssoftware.com/papers/HackproofingMySQL.pdf
[5] www.databasesecurity.com/informix/DatabaseHackersHandbook-AttackingInformix.pdf

Resources

Vendor
www.microsoft.com/sql/technologies/security/default.mspx
www.oracle.com/technology/deploy/security/index.html
www.ibm.com/security
www.postgresql.org/support/security.html
dev.mysql.com/doc/refman/6.0/en/security.html
www.sybase.com/

Database security sites

www.databasesecurity.com/
www.sqlsecurity.com/

Books

Cryptography in the Database, Kevin Keenan
The Database Hacker's Handbook, David Litchfield
Database Security and Auditing: Protecting Data Integrity and Accessibility, Sam
Effective Oracle Database 10g Security by Design, David Knox
Implementing Database Security and Auditing, Ron Ben Natan
Afyouni
Mastering SQL Server 2000 Security, Mike Young
SQL Server Security Distilled, Morris Lewis
Mysql Security Handbook, Wrox
The Oracle Hacker's Handbook, David Litchfield
Oracle Security, William Heney
Oracle Security Handbook, Marlene Theriault
SQL Server Security, Chip Andrews
Understanding DB2 9 Security, Rebecca Bond

Database Vulnerability Assessment Tools

NGSSQuirreL, www.ngssoftware.com/
AppDetective, www.appsecinc.com/
Scuba, www.imperva.com/