



<http://www.progressivebusinesstechnologytraining.com/>

Microsoft®  
**SQL Server** 2005

## **SQL Server 2005 Security Best Practices - Operational and Administrative Tasks**

SQL Server Technical Article

Writers: [Bob Beauchemin](#), [SQLskills](#)

Technical Reviewers: [Laurentiu Cristofor](#), [Al Comeau](#), [Sameer Tejani](#), [Devendra Tiwari](#),  
[Rob Walters](#), [Niraj Nagrani](#)

Published: [March 2007](#)

Applies To: SQL Server 2005 SP2

**Summary:** Security is a crucial part of any mission-critical application. This paper describes best practices for setting up and maintaining security in SQL Server 2005.

## Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. [MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.](#)

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, email address, logo, person, place or event is intended or should be inferred.

© 2007 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

## Table of Contents

<b>Introduction</b> .....	<b>4</b>
<b>Surface Area Reduction</b> .....	<b>4</b>
<b>Service Account Selection and Management</b> .....	<b>6</b>
<b>Authentication Mode</b> .....	<b>8</b>
<b>Network Connectivity</b> .....	<b>9</b>
<b>Lockdown of System Stored Procedures</b> .....	<b>13</b>
<b>Password Policy</b> .....	<b>14</b>
<b>Administrator Privileges</b> .....	<b>15</b>
<b>Database Ownership and Trust</b> .....	<b>17</b>
<b>Schemas</b> .....	<b>18</b>
<b>Authorization</b> .....	<b>19</b>
<b>Catalog Security</b> .....	<b>21</b>
<b>Remote Data Source Execution</b> .....	<b>21</b>
<b>Execution Context</b> .....	<b>22</b>
<b>Encryption</b> .....	<b>24</b>
<b>Auditing</b> .....	<b>26</b>
<b>Microsoft Baseline Security Analyzer and SQL Server Best Practices Analyzer</b>	<b>28</b>
<b>Patching</b> .....	<b>28</b>
<b>Conclusion</b> .....	<b>28</b>

## Introduction

This white paper covers some of the operational and administrative tasks associated with Microsoft® SQL Server™ 2005 security and enumerates best practices and operational and administrative tasks that will result in a more secure SQL Server system. Each topic describes a feature and best practices. For additional information on the specifics of utilities, features, and DDL statements referenced in this white paper, see SQL Server 2005 Books Online. Features and options that are new or defaults that are changed for SQL Server 2005 are identified. Coding examples for operational tasks use Transact-SQL, so understanding Transact-SQL is required for you to get the most out of this paper.

## Surface Area Reduction

SQL Server 2005 installation minimizes the "attack surface" because by default, optional features are not installed. During installation the administrator can choose to install:

- Database Engine
- Analysis Services Engine
- Reporting Services
- Integration Services
- Notification Services
- Documentation and Samples

It is a good practice to review which product features you actually need and install only those features. Later, install additional features only as needed. SQL Server 2005 includes sample databases for OLTP, data warehousing, and Analysis Services. Install sample databases on test servers only; they are not installed by default when you install the corresponding engine feature. SQL Server 2005 includes sample code covering every feature of the product. These samples are not installed by default and should be installed only on a development server, not on a production server. Each item of sample code has undergone a review to ensure that the code follows best practices for security. Each sample uses Microsoft Windows® security principals and illustrates the principal of least privilege.

SQL Server has always been a feature-rich database and the number of new features in SQL Server 2005 can be overwhelming. One way to make a system more secure is to limit the number of optional features that are installed and enabled by default. It is easier to enable features when they are needed than it is to enable everything by default and then turn off features that you do not need. This is the installation policy of SQL Server 2005, known as "off by default, enable when needed." One way to ensure that security policies are followed is to make secure settings the default and make them easy to use.

SQL Server 2005 provides a "one-stop" utility that can be used to enable optional features on a per-service and per-instance basis as needed. Although there are other utilities (such as Services in Control Panel), server configuration commands (such as **sp\_configure**), and APIs such as WMI (Windows Management Instrumentation) that you can use, the SQL Server Surface Area Configuration tool combines this functionality into a single utility program. This program can be used either from the command line or via a graphic user interface.

SQL Server Service Area Configuration divides configuration into two subsets: services and connections, and features. Use the Surface Area Configuration for Services and Connections tool to view the installed components of SQL Server and the client network interfaces for each engine component. The startup type for each service (Automatic, Manual, or Disabled)

and the client network interfaces that are available can be configured on a per-instance basis. Use the Surface Area Configuration for Features tool to view and configure instance-level features.

The features enabled for configuration are:

- CLR Integration
- Remote use of a dedicated administrator connection
- OLE Automation system procedures
- System procedures for Database Mail and SQL Mail
- Ad hoc remote queries (the OPENROWSET and OPENDATASOURCE functions)
- SQL Server Web Assistant
- **xp\_cmdshell** availability

The features enabled for viewing are:

- HTTP endpoints
- Service Broker endpoint

The SQL Server Surface Area Configuration command-line interface, `sac.exe`, permits you to import and export settings. This enables you to standardize the configuration of a group of SQL Server 2005 instances. You can import and export settings on a per-instance basis and also on a per-service basis by using command-line parameters. For a list of command-line parameters, use the `-?` command-line option. You must have **sysadmin** privilege to use this utility. The following code is an example of exporting all settings from the default instance of SQL Server on `server1` and importing them into `server2`:

```
sac out server1.out -S server1 -U admin -I MSSQLSERVER
sac in server1.out -S server2
```

When you upgrade an instance of SQL Server to SQL Server 2005 by performing an in-place upgrade, the configuration options of the instance are unchanged. Use SQL Server Surface Area Configuration to review feature usage and turn off features that are not needed. You can turn off the features in SQL Server Surface Area Configuration or by using the system stored procedure, **sp\_configure**. Here is an example of using **sp\_configure** to disallow the execution of **xp\_cmdshell** on a SQL Server instance:

```
-- Allow advanced options to be changed.
EXEC sp_configure 'show advanced options', 1
GO
-- Update the currently configured value for advanced options.
RECONFIGURE
GO
-- Disable the feature.
EXEC sp_configure 'xp_cmdshell', 0
GO
-- Update the currently configured value for this feature.
RECONFIGURE
```

GO

In SQL Server 2005, SQL Server Browser functionality has been factored into its own service and is no longer part of the core database engine. Additional functions are also factored into separate services. Services that are not a part of the core database engine and can be enabled or disabled separately include:

- SQL Server Active Directory Helper
- SQL Server Agent
- SQL Server FullText Search
- SQL Server Browser
- SQL Server VSS Writer

The SQL Server Browser service needs to be running only to connect to named SQL Server instances that use TCP/IP dynamic port assignments. It is not necessary to connect to default instances of SQL Server 2005 and named instances that use static TCP/IP ports. For a more secure configuration, always use static TCP/IP port assignments and disable the SQL Server Browser service. The VSS Writer allows backup and restore using the Volume Shadow Copy framework. This service is disabled by default. If you do not use Volume Shadow Copy, disable this service. If you are running SQL Server outside of an Active Directory® directory service, disable the Active Directory Helper.

#### **Best practices for surface area reduction**

- Install only those components that you will immediately use. Additional components can always be installed as needed.
- Enable only the optional features that you will immediately use.
- Review optional feature usage before doing an in-place upgrade and disable unneeded features either before or after the upgrade.
- Develop a policy with respect to permitted network connectivity choices. Use SQL Server Surface Area Configuration to standardize this policy.
- Develop a policy for the usage of optional features. Use SQL Server Surface Area Configuration to standardize optional feature enabling. Document any exceptions to the policy on a per-instance basis.
- Turn off unneeded services by setting the service to either Manual startup or Disabled.

#### ***Service Account Selection and Management***

SQL Server 2005 executes as a set of Windows services. Each service can be configured to use its own service account. This facility is exposed at installation. SQL Server provides a special tool, SQL Server Configuration Manager, to manage these accounts. In addition, these accounts can be set programmatically through the SQL Server WMI Provider for Configuration. When you select a Windows account to be a SQL Server service account, you have a choice of:

- Domain user that is not a Windows administrator
- Local user that is not a Windows administrator
- Network Service account
- Local System account
- Local user that is a Windows administrator
- Domain user that is a Windows administrator

When choosing service accounts, consider the principle of least privilege. The service account should have exactly the privileges that it needs to do its job and no more privileges. You also need to consider account isolation; the service accounts should not only be different from one another, they should not be used by any other service on the same server. Only the first two account types in the list above have both of these properties. Making the SQL Server service account an administrator, at either a server level or a domain level, bestows too many unneeded privileges and should never be done. The Local System account is not only an account with too many privileges, but it is a shared account and might be used by other services on the same server. Any other service that uses this account has the same set up privileges as the SQL Server service that uses the account. Although Network Service has network access and is not a Windows superuser account, it is a shareable account. This account is useable as a SQL Server service account only if you can ensure that no other services that use this account are installed on the server.

Using a local user or domain user that is not a Windows administrator is the best choice. If the server that is running SQL Server is part of a domain and must access domain resources such as file shares or uses linked server connections to other computers running SQL Server, a domain account is the best choice. If the server is not part of a domain (for example, a server running in the perimeter network (also known as the DMZ) in a Web application) or does not need to access domain resources, a local user that is not a Windows administrator is preferred.

Creating the user account that will be used as a SQL Server service account is easier in SQL Server 2005 than in previous versions. When SQL Server 2005 is installed, a Windows group is created for each SQL Server service, and the service account is placed in the appropriate group. To create a user that will serve as a SQL Server service account, simply create an "ordinary" account that is either a member of the Users group (non-domain user) or Domain Users group (domain user). During installation, the user is automatically placed in the SQL Server service group and the group is granted exactly the privileges that are needed.

If the service account needs additional privileges, the privilege should be granted to the appropriate Windows group, rather than granted directly to the service user account. This is consistent with the way access control lists are best managed in Windows in general. For example, the ability to use the SQL Server Instant File Initialization feature requires that the Perform Volume Maintenance Tasks user rights be set in the Group Policy Administration tool. This privilege should be granted to SQLServer2005MSSQLUser\$MachineName\$MSSQLSERVER group for the default instance of SQL Server on server "MachineName."

SQL Server service accounts should be changed only by using SQL Server Configuration Manager, or by using the equivalent functionality in the WMI APIs. Using Configuration Manager ensures that the new service account is placed in the appropriate Windows group, and is thus granted exactly the correct privileges to run the service. In addition, using SQL Server Configuration Manager also re-encrypts the service master key that is using the new account. For more information on the service master key, see [Encryption](#) later in this paper. Because SQL Server service accounts also abide by Windows password expiration policies, it is necessary to change the service account passwords at regular intervals. In SQL Server 2005, it is easier to abide by password expiration policies because changing the password of the service account does not require restarting SQL Server.

SQL Server 2005 requires that the service account have less privilege than in previous versions. Specifically, the privilege Act As Part of the Operating System (SE\_TCB\_NAME) is

not required for the service account unless SQL Server 2005 is running on the Microsoft Windows Server™ 2000 SP4 operating system. After doing an upgrade in place, use the Group Policy Administration tool to remove this privilege.

The SQL Server Agent service account requires **sysadmin** privilege in the SQL Server instance that it is associated with. In SQL Server 2005, SQL Server Agent job steps can be configured to use proxies that encapsulate alternate credentials. A CREDENTIAL is simply a database object that is a symbolic name for a Windows user and password. A single CREDENTIAL can be used with multiple SQL Server Agent proxies. To accommodate the principal of least privilege, do not give excessive privileges to the SQL Server Agent service account. Instead, use a proxy that corresponds to a CREDENTIAL that has just enough privilege to perform the required task. A CREDENTIAL can also be used to reduce the privilege for a specific task if the SQL Server Agent service account has been configured with more privileges than needed for the task. Proxies can be used for:

- ActiveX scripting
- Operating system (CmdExec)
- Replication agents
- Analysis Services commands and queries
- SSIS package execution (including maintenance plans)

#### **Best practices for SQL Server service accounts**

- Use a specific user account or domain account rather than a shared account for SQL Server services.
- Use a separate account for each service.
- Do not give any special privileges to the SQL Server service account; they will be assigned by group membership.
- Manage privileges through the SQL Server supplied group account rather than through individual service user accounts.
- Always use SQL Server Configuration Manager to change service accounts.
- Change the service account password at regular intervals.
- Use CREDENTIALS to execute job steps that require specific privileges rather than adjusting the privilege to the SQL Server Agent service account.
- If an agent user needs to execute a job that requires different Windows credentials, assign them a proxy account that has just enough permissions to get the task done.

#### ***Authentication Mode***

SQL Server has two authentication modes: Windows Authentication and Mixed Mode Authentication. In Windows Authentication mode, specific Windows user and group accounts are trusted to log in to SQL Server. Windows credentials are used in the process; that is, either NTLM or Kerberos credentials. Windows accounts use a series of encrypted messages to authenticate to SQL Server; no passwords are passed across the network during the authentication process. In Mixed Mode Authentication, both Windows accounts and SQL Server-specific accounts (known as SQL logins) are permitted. When SQL logins are used, SQL login passwords are passed across the network for authentication. This makes SQL logins less secure than Windows logins.

It is a best practice to use only Windows logins whenever possible. Using Windows logins with SQL Server achieves single sign-on and simplifies login administration. Password management uses the ordinary Windows password policies and password change APIs.

Users, groups, and passwords are managed by system administrators; SQL Server database administrators are only concerned with which users and groups are allowed access to SQL Server and with authorization management.

SQL logins should be confined to legacy applications, mostly in cases where the application is purchased from a third-party vendor and the authentication cannot be changed. Another use for SQL logins is with cross-platform client-server applications in which the non-Windows clients do not possess Windows logins. Although using SQL logins is discouraged, there are security improvements for SQL logins in SQL Server 2005. These improvements include the ability to have SQL logins use the password policy of the underlying operating system and better encryption when SQL passwords are passed over the network. We'll discuss each of these later in the paper.

SQL Server 2005 uses standard DDL statements to create both Windows logins and SQL logins. Using the CREATE LOGIN statement is preferred; the **sp\_addlogin** and **sp\_grantlogin** system stored procedures are supported for backward compatibility only. SQL Server 2005 also provides the ability to disable a login or change a login name by using the ALTER LOGIN DDL statement. For example, if you install SQL Server 2005 in Windows Authentication mode rather than Mixed Mode, the **sa** login is disabled. Use ALTER LOGIN rather than the procedures **sp\_denylogin** or **sp\_revokellogin**, which are supported for backward compatibility only.

If you install SQL Server in Windows Authentication mode, the **sa** login account is disabled and a random password is generated for it. If you later need to change to Mixed Mode Authentication and re-enable the **sa** login account, you will not know the password. Change the **sa** password to a known value after installation if you think you might ever need to use it.

### Best practices for authentication mode

- Always use Windows Authentication mode if possible.
- Use Mixed Mode Authentication only for legacy applications and non-Windows users.
- Use the standard login DDL statements instead of the compatibility system procedures.
- Change the **sa** account password to a known value if you might ever need to use it. Always use a strong password for the **sa** account and change the **sa** account password periodically.
- Do not manage SQL Server by using the **sa** login account; assign **sysadmin** privilege to a known user or group.
- Rename the **sa** account to a different account name to prevent attacks on the **sa** account by name.

### Network Connectivity

A standard network protocol is required to connect to the SQL Server database. There are no internal connections that bypass the network. SQL Server 2005 introduces an abstraction for managing any connectivity channel—entry points into a SQL Server instance are all represented as endpoints. Endpoints exist for the following network client connectivity protocols:

- Shared Memory
- Named Pipes
- TCP/IP
- VIA

- Dedicated administrator connection

In addition, endpoints may be defined to permit access to the SQL Server instance for:

- Service Broker
- HTTP Web Services
- Database mirroring

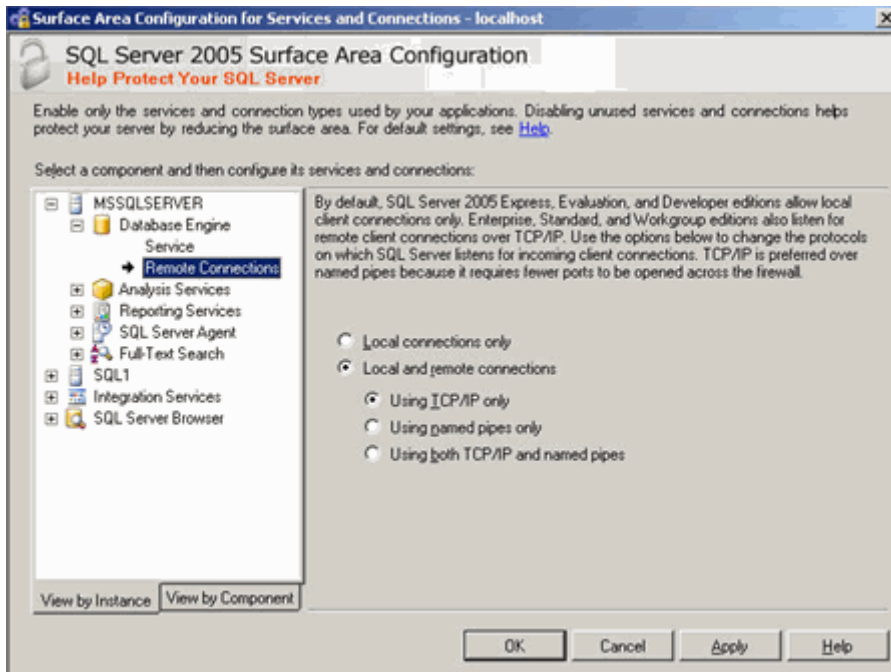
Following is an example of creating an endpoint for Service Broker.

```
CREATE ENDPOINT BrokerEndpoint_SQLDEV01
AS TCP
    ( LISTENER_PORT = 4022 )
FOR SERVICE_BROKER
    ( AUTHENTICATION = WINDOWS )
```

SQL Server 2005 discontinues support for some network protocols that were available with earlier versions of SQL Server, including IPX/SPX, Appletalk, and Banyon Vines.

In keeping with the general policy of "off by default, enable only when needed," no Service Broker, HTTP, or database mirroring endpoints are created when SQL Server 2005 is installed, and the VIA endpoint is disabled by default. In addition, in SQL Server 2005 Express Edition, SQL Server 2005 Developer Edition, and SQL Server 2005 Evaluation Edition, the Named Pipes and TCP/IP protocols are disabled by default. Only Shared Memory is available by default in those editions. The dedicated administrator connection (DAC), new with SQL Server 2005, is available only locally by default, although it can be made available remotely. Note that the DAC is not available in SQL Server Express Edition by default and requires that the server be run with a special trace flag to enable it. Access to database endpoints requires the login principal to have CONNECT permission. By default, no login account has CONNECT permission to Service Broker or HTTP Web Services endpoints. This restricts access paths and blocks some known attack vectors. It is a best practice to enable only those protocols that are needed. For example, if TCP/IP is sufficient, there is no need to enable the Named Pipes protocol.

Although endpoint administration can be accomplished via DDL, the administration process is made easier and policy can be made more uniform by using the SQL Server Surface Area Configuration tool and SQL Server Configuration Manager. SQL Server Surface Area Configuration provides a simplified user interface for enabling or disabling client protocols for a SQL Server instance, as shown in Figure 1 and Figure 2. Configuration is described in Knowledge Base article KB914277, [How to configure SQL Server 2005 to allow remote connections](#), as well as in SQL Server 2005 Books Online. A screenshot showing the remote connections configuration dialog box is shown in Figure 1.



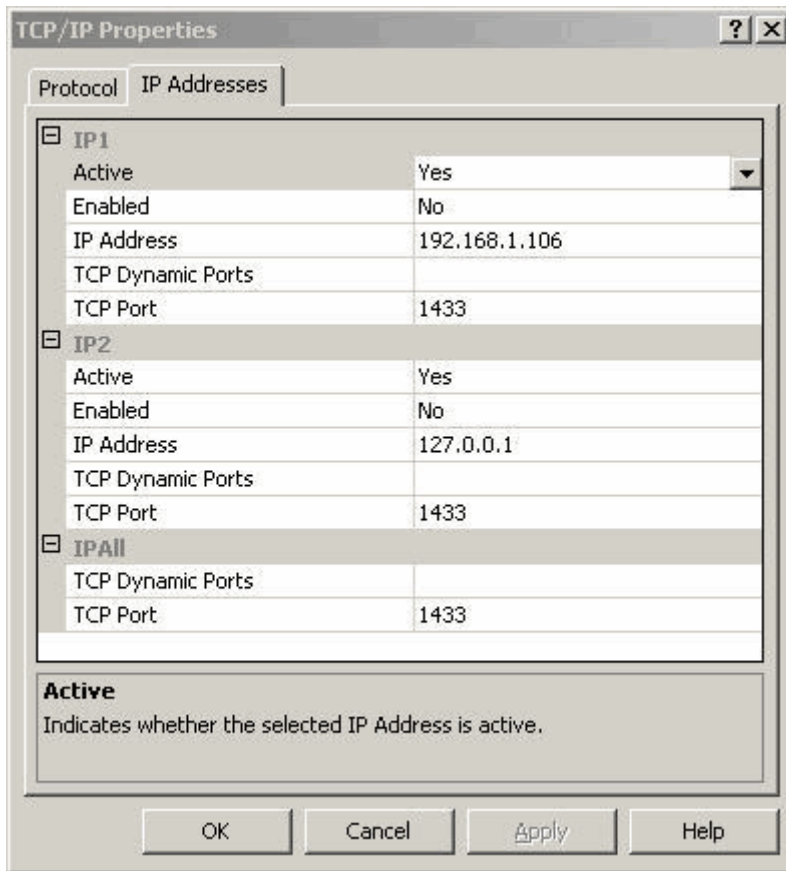
**Figure 1 Configuring remote connections**

In the Surface Area Configuration for Services and Connections dialog box, you can see if any HTTP or Service Broker endpoints are defined for the instance. New endpoints must be defined by using DDL statements; SQL Server Surface Area Configuration cannot be used to define these. You can use the Surface Area Configuration for Features tool to enable remote access to the dedicated administrator connection.

SQL Server Configuration Manager provides more granular configuration of server protocols. With Configuration Manager, you can:

- Choose a certificate for SSL encryption.
- Allow only encryption connections from clients.
- Hide an instance of SQL Server from the server enumeration APIs.
- Enable and disable TCP/IP, Shared Memory, Named Pipes, and VIA protocols.
- Configure the name of the pipe each instance of SQL Server will use.
- Configure a TCP/IP port number that each instance listens on for TCP/IP connections.
- Choose whether to use TCP/IP dynamic port assignment for named instances.

The dialog for configuring TCP/IP address properties such as port numbers and dynamic port assignment is shown in Figure 2.



**Figure 2 TCP/IP Addresses configuration page in SQL Server Configuration Manager**

SQL Server 2005 can use an encrypted channel for two reasons: to encrypt credentials for SQL logins, and to provide end-to-end encryption of entire sessions. Using encrypted sessions requires using a client API that supports these. The OLE DB, ODBC, and ADO.NET clients all support encrypted sessions; currently the Microsoft JDBC client does not. The other reason for using SSL is to encrypt credentials during the login process for SQL logins when a password is passed across the network. If an SSL certificate is installed in a SQL Server instance, that certificate is used for credential encryption. If an SSL certificate is not installed, SQL Server 2005 can generate a self-signed certificate and use this certificate instead. Using the self-signed certificate prevents passive man-in-the-middle attacks, in which the man-in-the-middle intercepts network traffic, but does not provide mutual authentication. Using an SSL certificate with a trusted root certificate authority prevents active man-in-the-middle attacks and provides mutual authentication.

In SQL Server 2005, you can GRANT, REVOKE, or DENY permission to CONNECT to a specific endpoint on a per-login basis. By default, all logins are GRANTED permission on the Shared Memory, Named Pipes, TCP/IP, and VIA endpoints. You must specifically GRANT users CONNECT permission to other endpoints; no users are GRANTED this privilege by default. An example of granting this permission is:

```
GRANT CONNECT ON MyHTTPEndpoint TO MyDomain\Accounting
```

### Best practices for network connectivity

- Limit the network protocols supported.
- Do not enable network protocols unless they are needed.
- Do not expose a server that is running SQL Server to the public Internet.
- Configure named instances of SQL Server to use specific port assignments for TCP/IP rather than dynamic ports.
- If you must support SQL logins, install an SSL certificate from a trusted certificate authority rather than using SQL Server 2005 self-signed certificates.
- Use "allow only encrypted connections" only if needed for end-to-end encryption of sensitive sessions.
- Grant CONNECT permission only on endpoints to logins that need to use them. Explicitly deny CONNECT permission to endpoints that are not needed by users or groups.

### *Lockdown of System Stored Procedures*

SQL Server uses system stored procedures to accomplish some administrative tasks. These procedures almost always begin with the prefix **xp\_** or **sp\_**. Even with the introduction of standard DDL for some tasks (for example, creating logins and users), system procedures remain the only way to accomplish tasks such as sending mail or invoking COM components. System extended stored procedures in particular are used to access resources outside the SQL Server instance. Most system stored procedures contain the relevant security checks as part of the procedure and also perform impersonation so that they run as the Windows login that invoked the procedure. An example of this is **sp\_reserve\_http\_namespace**, which impersonates the current login and then attempts to reserve part of the HTTP namespace (HTTP.SYS) by using a low-level operating system function.

Because some system procedures interact with the operating system or execute code outside of the normal SQL Server permissions, they can constitute a security risk. System stored procedures such as **xp\_cmdshell** or **sp\_send\_dbmail** are off by default and should remain disabled unless there is a reason to use them. In SQL Server 2005, you no longer need to use stored procedures that access the underlying operating system or network outside of the SQL Server permission space. SQLCLR procedures executing in EXTERNAL\_ACCESS mode are subject to SQL Server permissions, and SQLCLR procedures executing in UNSAFE mode are subject to some, but not all, security checks. For example, to catalog a SQLCLR assembly categorized as EXTERNAL\_ACCESS or UNSAFE, either the database must be marked as TRUSTWORTHY (see [Database Ownership and Trust](#)) or the assembly must be signed with a certificate or asymmetric key that is cataloged to the **master** database. SQLCLR procedures should replace user-written extended stored procedures in the future.

Some categories of system stored procedures can be managed by using SQL Server Surface Area Configuration. These include:

- **xp\_cmdshell** - executes a command in the underlying operating system
- Database Mail procedures
- SQL Mail procedures
- COM component procedures (e.g. sp\_OACreate)

Enable these procedures only if necessary.

Some system stored procedures, such as procedures that use SQLDMO and SQLSMO libraries, cannot be configured by using SQL Server Surface Area Configuration. They must

be configured by using **sp\_configure** or SSMS directly. SSMS or **sp\_configure** can also be used to set most of the configuration feature settings that are set by using SQL Server Surface Area Configuration.

The system stored procedures should not be dropped from the database; dropping these can cause problems when applying service packs. Removing the system stored procedures results in an unsupported configuration. It is usually unnecessary to completely DENY all users access to the system stored procedures, as these stored procedures have the appropriate permission checks internal to the procedure as well as external.

### Best practices for system stored procedures

- Disable **xp\_cmdshell** unless it is absolutely needed.
- Disable COM components once all COM components have been converted to SQLCLR.
- Disable both mail procedures (Database Mail and SQL Mail) unless you need to send mail from SQL Server. Prefer Database Mail as soon as you can convert to it.
- Use SQL Server Surface Area Configuration to enforce a standard policy for extended procedure usage.
- Document each exception to the standard policy.
- Do not remove the system stored procedures by dropping them.
- Do not DENY all users/administrators access to the extended procedures.

### Password Policy

Windows logins abide by the login policies of the underlying operating system. These policies can be set using the Domain Security Policy or Local Security Policy administrator Control Panel applets. Login policies fall into two categories: Password policies and Account Lockout policies. Password policies include:

- Enforce Password History
- Minimum and Maximum Password Age
- Minimum Password Length
- Password Must Meet Complexity Requirements
- Passwords are Stored Using Reversible Encryption (Note: this setting does not apply to SQL Server)

Account Lockout policies include:

- Account Lockout Threshold (Number of invalid logins before lockout)
- Account Lockout Duration (Amount of time locked out)
- Reset Lockout Counter After *n* Minutes

In SQL Server 2005, SQL logins can also go by the login policies of the underlying operating system if the operating system supports it. The operating system must support the system call **NetValidatePasswordPolicy**. Currently, the only operating system that supports this is Windows Server 2003 and later versions. If you use SQL logins, run SQL Server 2005 on a Windows Server 2003 or later operating system. CREATE LOGIN parameters determine whether the login goes by the operating system policies. These parameters are:

- CHECK\_POLICY
- CHECK\_EXPIRATION
- MUST\_CHANGE

CHECK\_POLICY specifies that the SQL login must abide by the Windows login policies and Account Lockout policies, with the exception of password expiration. This is because, if SQL logins must go by the Windows password expiration policy, underlying applications must be outfitted with a mechanism for password changing. Most applications currently do not provide a way to change SQL login passwords. In SQL Server 2005, both SSMS and SQLCMD provide a way to change SQL Server passwords for SQL logins. Consider outfitting your applications with a password-changing mechanism as soon as possible. Having built-in password changing also allows logins to be created with the MUST\_CHANGE parameter; using this parameter requires the user to change the password at the time of the first login. Administrators should be aware of the fact that password length and complexity policies, but not expiration policies, apply to passwords used with encryption keys as well as to passwords used with SQL logins. For a description of encryption keys, see [Encryption](#). When SQL logins are used on pre-Windows 2003 operating systems, there is a series of hard-coded password policies in lieu of the domain or operating system policies if CHECK\_POLICY = ON. These policies are enumerated in SQL Server Books Online.

### Best practices for password policy

- Mandate a strong password policy, including an expiration and a complexity policy for your organization.
- If you must use SQL logins, ensure that SQL Server 2005 runs on the Windows Server 2003 operating system and use password policies.
- Outfit your applications with a mechanism to change SQL login passwords.
- Set MUST\_CHANGE for new logins.

### Administrator Privileges

SQL Server 2005 makes all permissions grantable and also makes grantable permissions more granular than in previous versions. Privileges with elevated permissions now include:

- Members of the **sysadmin** server role.
- The **sa** built-in login, if it is enabled.
- Any login with CONTROL SERVER permission.

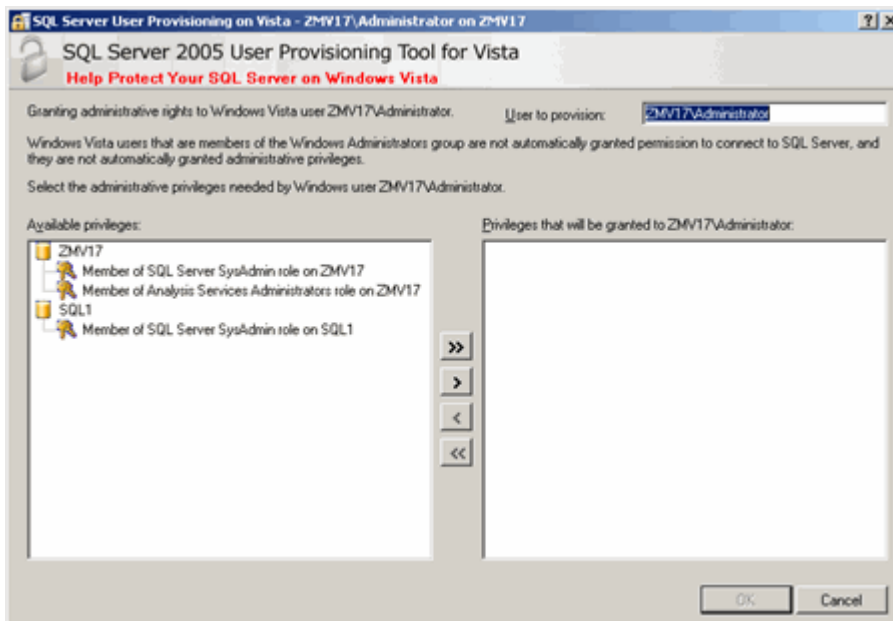
CONTROL SERVER permission is new in SQL Server 2005. Change your auditing procedures to include any login with CONTROL SERVER permission.

SQL Server automatically grants the server's Administrators group (BUILTIN\administrators) the **sysadmin** server role. When running SQL Server 2005 under Microsoft Windows Vista™, the operating system does not recognize membership in the BUILTIN\Administrators group unless the user has elevated themselves to a full administrator. In SP2, you can use SQL Server Surface Area Configuration to enable a principal to act as administrator by selecting **Add New Administrator** from the main window as shown in Figure 3.



**Figure 3 Adding a new administrator in SP2 SQL Server Surface Area Configuration**

Clicking on this link opens the SQL Server 2005 User Provisioning Tool for Vista as shown in Figure 4. This tool can also be automatically invoked as the last step of an SQL Server 2005 SP2 installation.



#### Figure 4 The SQL Server 2005 User Provisioning Tool for Vista

When running SQL Server Express SP2 under the Vista operating system, Set Up incorporates the specification of a specific principal to act as administrator. SQL Server Express SP2 Set Up also allows command-line options to turn user instances on or off (ENABLERANU) and to add the current Set Up user to the **SQL Server Administrator** role (ADDUSERASADMIN). For more detailed information, see [Configuration Options \(SQL Server Express\)](#) in SQL Server 2005 SP2 Books Online. For additional security-related considerations when running SQL Server 2005 with the Windows Vista operating system, see the [SQL Server 2005 SP2 Readme file](#). In particular, see section 5.5.2 "Issues Caused by User Account Control in Windows Vista."

For accountability in the database, avoid relying on the Administrators group and add only specific database administrators to the **sysadmin** role. Another option is to have a specific DatabaseAdministrators role at the operating system level. Minimizing the number of administrators who have **sysadmin** or CONTROL SERVER privilege also makes it easier to resolve problems; fewer logins with administrator privilege means fewer people to check with if things go wrong. The permission VIEW SERVER STATE is useful for allowing administrators and troubleshooters to view server information (dynamic management views) without granting full **sysadmin** or CONTROL SERVER permission.

#### Best practices for administrator privileges

- Use administrator privileges only when needed.
- Minimize the number of administrators.
- Provision admin principals explicitly.
- Have multiple distinct administrators if more than one is needed.
- Avoid dependency on the builtin\administrators Windows group.

#### Database Ownership and Trust

A SQL Server instance can contain multiple user databases. Each user database has a specific owner; the owner defaults to the database creator. By definition, members of the **sysadmin** server role (including system administrators if they have access to SQL Server through their default group account) are database owners (DBOs) in every user database. In addition, there is a database role, **db\_owner**, in every user database. Members of the **db\_owner** role have approximately the same privileges as the **dbo** user.

SQL Server can be thought of as running in two distinct modes, which can be referred to as *IT department mode* and *ISV mode*. These are not database settings but simply different ways to manage SQL Server. In an IT department, the **sysadmin** of the instance manages all user databases. In an Internet service provider environment (say, a Web-hosting service), each customer is permitted to manage their own database and is restricted from accessing system databases or other user databases. For example, the databases of two competing companies could be hosted by the same Internet service provider (ISV) and exist in the same SQL Server instance. Dangerous code could be added to a user database when attached to its original instance, and the code would be enabled on the ISV instance when deployed. This situation makes controlling cross-database access crucial.

If each database is owned and managed by the same general entity, it is still not a good practice to establish a "trust relationship" with a database unless an application-specific feature, such as cross-database Service Broker communication, is required. A trust relationship between databases can be established by allowing cross-database ownership

chaining or by marking a database as trusted by the instance by using the TRUSTWORTHY property. An example of setting the TRUSTWORTHY property follows:

```
ALTER DATABASE pubs SET TRUSTWORTHY ON
```

### Best practices for database ownership and trust

- Have distinct owners for databases; not all databases should be owned by **sa**.
- Minimize the number of owners for each database.
- Confer trust selectively.
- Leave the Cross-Database Ownership Chaining setting off unless multiple databases are deployed at a single unit.
- Migrate usage to selective trust instead of using the TRUSTWORTHY property.

### Schemas

SQL Server 2005 introduces schemas to the database. A *schema* is simply a named container for database objects. Each schema is a scope that fits into the hierarchy between database level and object level, and each schema has a specific owner. The owner of a schema can be a user, a database role, or an application role. The schema name takes the place of the owner name in the SQL Server multi-part object naming scheme. In SQL Server 2000 and previous versions, a table named Employee that was part of a database named Payroll and was owned by a user name Bob would be payroll.bob.employee. In SQL Server 2005, the table would have to be part of a schema. If payroll\_app is the name of the SQL Server 2005 schema, the table name in SQL Server 2005 is payroll.payroll\_app.employee.

Schemas solve an administration problem that occurs when each database object is named after the user who creates it. In SQL Server versions prior to 2005, if a user named Bob (who is not **dbo**) creates a series of tables, the tables would be named after Bob. If Bob leaves the company or changes job assignments, these tables would have to be manually transferred to another user. If this transfer were not performed, a security problem could ensue. Because of this, prior to SQL Server 2005, DBAs were unlikely to allow individual users to create database objects such as tables. Each table would be created by someone acting as the special **dbo** user and would have a user name of **dbo**. Because, in SQL Server 2005, schemas can be owned by roles, special roles can be created to own schemas if needed—every database object need not be owned by **dbo**. Not having every object owned by **dbo** makes for more granular object management and makes it possible for users (or applications) that need to dynamically create tables to do so without **dbo** permission.

Having schemas that are role-based does not mean that it's a good practice to have every user be a schema owner. Only users who need to create database objects should be permitted to do so. The ability to create objects does not imply schema ownership; GRANTing Bob ALTER SCHEMA permission in the payroll\_app schema can be accomplished without making Bob a schema owner. In addition, granting CREATE TABLE to a user does not allow that user to create tables; the user must also have ALTER SCHEMA permission on some schema in order to have a schema in which to create the table. Objects created in a schema are owned by the schema owner by default, not by the creator of the object. This makes it possible for a user to create tables in a known schema without the administrative problems that ensue when that user leaves the company or switches job assignments.

Each user has a default schema. If an object is created or referenced in a SQL statement by using a one-part name, SQL Server first looks in the user's default schema. If the object isn't found there, SQL Server looks in the **dbo** schema. The user's default schema is assigned by using the CREATE USER or ALTER USER DDL statements. If the default schema is specified, the default is **dbo**. Using named schemas for like groups of database objects and assigning each user's default schema to **dbo** is a way to mandate using two-part object names in SQL statements. This is because objects that are not in the **dbo** schema will not be found when a one-part object name is specified. Migrating groups of user objects out of the **dbo** schema is also a good way to allow users to create and manage objects if needed (for example, to install an application package) without making the installing user **dbo**.

### Best practices for using schemas

- Group like objects together into the same schema.
- Manage database object security by using ownership and permissions at the schema level.
- Have distinct owners for schemas.
- Not all schemas should be owned by **dbo**.
- Minimize the number of owners for each schema.

### Authorization

Authorization is the process of granting permissions on securables to users. At an operating system level, securables might be files, directories, registry keys, or shared printers. In SQL Server, securables are database objects. SQL Server principals include both instance-level principals, such as Windows logins, Windows group logins, SQL Server logins, and server roles and database-level principals, such as users, database roles, and application roles. Except for a few objects that are instance-scoped, most database objects, such as tables, views, and procedures are schema-scoped. This means that authorization is usually granted to database-level principals.

In SQL Server, authorization is accomplished via Data Access Language (DAL) rather than DDL or DML. In addition to the two DAL verbs, GRANT and REVOKE, mandated by the ISO-ANSI standard, SQL Server also contains a DENY DAL verb. DENY differs from REVOKE when a user is a member of more than one database principal. If a user Fred is a member of three database roles A, B, and C and roles A and B are GRANTED permission to a securable, if the permission is REVOKEd from role C, Fred still can access the securable. If the securable is DENYed to role C, Fred cannot access the securable. This makes managing SQL Server similar to managing other parts of the Windows family of operating systems.

SQL Server 2005 makes each securable available by using DAL statements and makes permissions more granular than in previous versions. For example, in SQL Server 2000 and earlier versions, certain functions were available only if a login was part of the **sysadmin** role. Now **sysadmin** role permissions are defined in terms of GRANTs. Equivalent access to securables can be achieved by GRANTing a login the CONTROL SERVER permission.

An example of better granularity is the ability to use SQL Server Profiler to trace events in a particular database. In SQL Server 2000, this ability was limited to the special **dbo** user. The new granular permissions are also arranged in a hierarchy; some permissions imply other permissions. For example, CONTROL permission on a database object type implies ALTER permission on that object as well as all other object-level permissions.

SQL Server 2005 also introduces the concept of granting permissions on all of the objects in a schema. ALTER permission on a SCHEMA includes the ability to CREATE, ALTER, or DROP

objects in that SCHEMA. The DAL statement that grants access to all securables in the payroll schema is:

```
GRANT SELECT ON schema::payroll TO fred
```

The advantage of granting permissions at the schema level is that the user automatically has permissions on all new objects created in the schema; explicit grant after object creation is not needed. For more information on the permission hierarchy, see the Permission Hierarchy section of SQL Server Books Online.

A best practice for authorization is to encapsulate access through modules such as stored procedures and user-defined functions. Hiding access behind procedural code means that users can only access objects in the way the developer and database administrator (DBA) intend; ad hoc changes to objects are disallowed. An example of this technique would be permitting access to the employee pay rate table only through a stored procedure "UpdatePayRate." Users that need to update pay rates would be granted EXECUTE access to the procedure, rather than UPDATE access to the table itself. In SQL Server 2000 and earlier versions, encapsulating access was dependent on a SQL Server feature known as *ownership chains*. In an ownership chain, if the owner of stored procedure A and the owner of table B that the stored procedure accesses are the same, no permission check is done. Although this works well most of the time, even with multiple levels of stored procedures, ownership chains do not work when:

- The database objects are in two different databases (unless cross-database ownership chaining is enabled).
- The procedure uses dynamic SQL.
- The procedure is a SQLCLR procedure.

SQL Server 2005 contains features to address these shortcomings, including signing of procedural code, alternate execution context, and a TRUSTWORTHY database property if ownership chaining is desirable because a single application encompasses multiple databases. All of these features are discussed in this white paper.

A login can only be granted authorization to objects in a database if a database user has been mapped to the login. A special user, **guest**, exists to permit access to a database for logins that are not mapped to a specific database user. Because any login can use the database through the **guest** user, it is suggested that the **guest** user not be enabled.

SQL Server 2005 contains a new type of user, a user that is not mapped to a login. Users that are not mapped to logins provide an alternative to using application roles. You can invoke selective impersonation by using the EXECUTE AS statement (see [Execution Context](#) later in this paper) and allow that user only the privileges needed to perform a specific task. Using users without logins makes it easier to move the application to a new instance and limits the connectivity requirements for the function. You create a user without a login using DDL:

```
CREATE USER mynewuser WITHOUT LOGIN
```

### Best practices for database object authorization

- Encapsulate access within modules.
- Manage permissions via database roles or Windows groups.
- Use permission granularity to implement the principle of least privilege.

- Do not enable **guest** access.
- Use users without logins instead of application roles

### *Catalog Security*

Information about databases, tables, and other database objects is kept in the system catalog. The system metadata exists in tables in the **master** database and in user databases. These metadata tables are exposed through metadata views. In SQL Server 2000, the system catalog was publicly readable and, the instance could be configured to make the system tables writeable as well. In SQL Server 2005, the system metadata tables are read-only and their structure has changed considerably. The only way that the system metadata tables are readable at all is in single-user mode. Also in SQL Server 2005, the system metadata views were refactored and made part of a special schema, the **sys** schema. So as not to break existing applications, a set of compatibility metadata views are exposed. The compatibility views may be removed in a future release of SQL Server.

SQL Server 2005 makes all metadata views secured by default. This includes:

- The new metadata views (for example, **sys.tables**, **sys.procedures**).
- The compatibility metadata views (for example, **sysindexes**, **sysobjects**).
- The INFORMATION\_SCHEMA views (provided for SQL-92 compliance).

The information in the system metadata views is secured on a per-row basis. In order to be able to see system metadata for an object, a user must have some permission on the object. For example, to see metadata about the `dbo.authors` table, `SELECT` permission on the table is sufficient. This prohibits browsing the system catalog by users who do not have appropriate object access. Discovery is often the first level of prevention. There are two exceptions to this rule: **sys.databases** and **sys.schemas** are public-readable. These metadata views may be secured with the `DENY` verb if required.

Some applications present lists of database objects to the user through a graphic user interface. It may be necessary to keep the user interface the same by permitting users to view information about database objects while giving them no other explicit permission on the object. A special permission, `VIEW DEFINITION`, exists for this purpose.

#### **Best practices for catalog security**

- The catalog views are secure by default. No additional action is required to secure them.
- Grant `VIEW DEFINITION` selectively at the object, schema, database, or server level to grant permission to view system metadata without conferring additional permissions.
- Review legacy applications that may depend on access to system metadata when migrating the applications to SQL Server 2005.

### *Remote Data Source Execution*

There are two ways that procedural code can be executed on a remote instance of SQL Server: configuring a linked server definition with the remote SQL Server and configuring a remote server definition for it. Remote servers are supported only for backward compatibility with earlier versions of SQL Server and should be phased out in preference to linked servers. Linked servers allow more granular security than remote servers. Ad hoc queries through linked servers (`OPENROWSET` and `OPENDATASOURCE`) are disabled by default in a newly installed instance of SQL Server 2005.

When you use Windows to authenticate to SQL Server, you are using a Windows network credential. Network credentials that use both NTLM and Kerberos security systems are valid for one network "hop" by default. If you use network credentials to log on to SQL Server and attempt to use the same credentials to connect via a linked server to a SQL Server instance on a different computer, the credentials will not be valid. This is known as the "double hop problem" and also occurs in environments that use Windows authentication to connect to a Web server and attempt to use impersonation to connect to SQL Server. If you use Kerberos for authentication, you can enable constrained delegation, that is, delegation of credentials constrained to a specific application, to overcome the "double hop problem." Only Kerberos authentication supports delegation of Windows credentials. For more information, see *Constrained Delegation in SQL Server Books Online*.

### **Best practices for remote data source execution**

- Phase out any remote server definitions.
- Replace remote servers with linked servers.
- Leave ad hoc queries through linked servers disabled unless they are absolutely needed.
- Use constrained delegation if pass-through authentication to a linked server is necessary.

### *Execution Context*

SQL Server always executes SQL statements and procedural code as the currently logged on user. This behavior is a SQL Server-specific behavior and is made possible, in the case of procedural code, by the concept of ownership chains. That is, although a stored procedure executes as the caller of the stored procedure rather than as the owner, if ownership chaining is in place, permissions are not checked for object access and stored procedures can be used to encapsulate tables, as mentioned previously in this paper. In SQL Server 2005, the creator of a procedure can declaratively set the execution context of the procedure by using the EXECUTE AS keyword in the CREATE PROCEDURE, FUNCTION, and TRIGGER statements. The execution context choices are:

- EXECUTE AS CALLER - the caller of the procedure (no impersonation). This is the only pre-SQL Server 2005 behavior.
- EXECUTE AS OWNER - the owner of the procedure.
- EXECUTE AS SELF - the creator of the procedure.
- EXECUTE AS 'username' - a specific user.

To maintain backward compatibility, EXECUTE AS CALLER is the default. The distinction between AS OWNER and AS SELF is needed because the creator of the procedure may not be the owner of the schema in which the procedure resides. In this case, AS SELF refers to the procedure owner, AS OWNER refers to the object owner (the schema owner). In order to use EXECUTE AS 'username', the procedure creator must have IMPERSONATE permission on the user named in the execution context.

One reason to use an alternate execution context would be when a procedure executes without a particular execution context. An example of this is a service broker queue activation procedure. In addition, EXECUTE AS OWNER can be used to circumvent problems that are caused when ownership chains are broken. For example, ownership chains in a procedure are always broken when dynamic SQL statements (such as **sp\_executeSQL**) are used.

Often what is needed is to grant the appropriate permissions to the procedural code itself, rather than either changing the execution context or relying on the caller's permissions. SQL Server 2005 offers a much more granular way of associating privileges with procedural code—code signing. By using the ADD SIGNATURE DDL statement, you can sign the procedure with a certificate or asymmetric key. A user can then be created for the certificate or asymmetric key itself and permissions assigned to that user. When the procedure is executed, the code executes with a combination of the caller's permissions and the key/certificate's permissions. An example of this would be:

```
CREATE CERTIFICATE HRCertificate
    WITH ENCRYPTION BY PASSWORD = 'HacdeNj162kqT'
CREATE USER HRCertificateUser
    FOR CERTIFICATE HRCertificate WITHOUT LOGIN
GRANT UPDATE ON pension_criteria TO HRCertificate
-- this gives the procedure update_pension_criteria
-- additional privileges of HRCertificate
ADD SIGNATURE TO update_pension_criteria BY CERTIFICATE HRCertificate
-- backup the private key and remove it from the certificate,
-- so that the procedure cannot be re-signed without permission
BACKUP CERTIFICATE HRCertificate
    TO FILE = 'c:\certs_backup\HRCertificate.cer'
    WITH PRIVATE KEY (FILE = 'c:\certs_backup\ HRCertificate.pvk',
        ENCRYPTION BY PASSWORD = 'jBjebfP43j1!',
        DECRYPTION BY PASSWORD = 'eWyveyYqW96A@!q')
ALTER CERTIFICATE HRCertificate REMOVE PRIVATE KEY
```

EXECUTE AS can also be used to set the execution context within an SQL batch. In this form, the SQL batch contains an EXECUTE AS USER='someuser' or EXECUTE AS LOGIN='somelgin' statement. This alternate execution context lasts until the REVERT statement is encountered. EXECUTE AS and REVERT blocks can also be nested; REVERT reverts one level of execution context. As with EXECUTE AS and procedural code, the user changing the execution context must have IMPERSONATE permission on the user or login being impersonated. EXECUTE AS in SQL batches should be used as a replacement for the SETUSER statement, which is much less flexible.

If the execution context is set but should not be reverted without permission, you can use EXECUTE AS ... WITH COOKIE or EXECUTE AS ... WITH NO REVERT. When WITH COOKIE is specified, a binary cookie is returned to the caller of EXECUTE AS and the cookie must be supplied in order to REVERT back to the original context.

When a procedure or batch uses an alternate execution context, the system functions normally used for auditing, such as SUSER\_NAME(), return the name of the impersonated user rather than the name of the original user or original login. A new system function, ORIGINAL\_LOGIN(), can be used to obtain the original login, regardless of the number of levels of impersonation used.

### Best practices for execution context

- Set execution context on modules explicitly rather than letting it default.
- Use EXECUTE AS instead of SETUSER.
- Use WITH NO REVERT/COOKIE instead of Application Roles.
- Consider using code signing of procedural code if a single granular additional privilege is required for the procedure.

### Encryption

SQL Server 2005 has built-in data encryption. The data encryption exists at a cell level and is accomplished by means of built-in system procedures. Encrypting data requires secure encryption keys and key management. A key management hierarchy is built into SQL Server 2005. Each instance of SQL Server has a built-in service master key that is generated at installation; specifically, the first time that SQL Server is started after installation. The service master key is encrypted by using both the SQL Server Service account key and also the machine key. Both encryptions use the DPAPI (Data Protection API). A database administrator can define a database master key by using the following DDL.

```
CREATE MASTER KEY
```

```
WITH ENCRYPTION BY PASSWORD = '87(HyfdlkRM?_764#GRtj*(NS£"_{^$( '
```

This key is actually encrypted and stored twice by default. Encryption that uses a password and storage in the database is required. Encryption that uses the service master key and storage in the **master** database is optional; it is useful to be able to automatically open the database master key without specifying the password. The service master key and database master keys can be backed up and restored separately from the rest of the database.

SQL Server 2005 can use DDL to define certificates, asymmetric keys, and symmetric keys on a per-database basis. Certificates and asymmetric keys consist of a private key/public key pair. The public key can be used to encrypt data that can be decrypted only by using the private key. Or, for the sake of performance, the public key can be used to encrypt a hash that can be decrypted only by using the private key. Encrypted checksum generation to ensure non-repudiation is known as *signing*.

Alternatively, the private key can be used to encrypt data that can be decrypted by the receiver by using the public key. A symmetric key consists of a single key that is used for encryption and decryption. Symmetric keys are generally used for data encryption because they are orders of magnitude faster than asymmetric keys for encryption and decryption. However, distributing symmetric keys can be difficult because both parties must have the same copy of the key. In addition, it is not possible with symmetric key encryption to determine which user encrypted the data. Asymmetric keys can be used to encrypt and decrypt data but ordinarily they are used to encrypt and decrypt symmetric keys; the symmetric keys are used for the data encryption. This is the preferred way to encrypt data for the best security and performance. Symmetric keys can also be protected by individual passwords.

SQL Server 2005 makes use of and also can generate X.509 certificates. A *certificate* is simply an asymmetric key pair with additional metadata, including a subject (the person the key is intended for), root certificate authority (who vouches for the certificate's authenticity), and expiration date. SQL Server generates self-signed certificates (SQL Server

itself is the root certificate authority) with a default expiration date of one year. The expiration date and subject can be specified in the DDL statement. SQL Server does not use certificate "negative lists" or the expiration date with data encryption. A certificate can be backed up and restored separately from the database; certificates, asymmetric keys, and symmetric keys are backed up with the database. A variety of block cipher encryption algorithms are supported, including DES, Triple DES, and AES (Rijndael) algorithms for symmetric keys and RSA for asymmetric keys. A variety of key strengths are supported for each algorithm. Stream cipher algorithms, such as RC4 are also supported but should NOT be used for data encryption. Some algorithms (such as AES) are not supported by all operating systems that can host SQL Server. User-defined algorithms are not supported. The key algorithm and key length choice should be predicated on the sensitivity of the data. SQL Server encrypts data on a cell level—data is specifically encrypted before it is stored into a column value and each row can use a different encryption key for a specific column. To use data encryption, a column must use the VARBINARY data type. The length of the column depends on the encryption algorithm used and the length of the data to be encrypted (see [Choosing an Encryption Algorithm](#) in SQL Server Books Online). The KEY\_GUID of the key that is used for encryption is stored with the column value. When the data is decrypted, this KEY\_GUID is checked against all open keys in the session. The data uses initialization vectors (also known as *salted hashes*). Because of this, it is not possible to determine if two values are identical by looking at the encrypted value. This means, for example, that I cannot determine all of the patients who have a diagnosis of Flu if I know that Patient A has a diagnosis of Flu. Although this means that data is more secure, it also means that you cannot use a column that is encrypted by using the data encryption routines in indexes, because data values are not comparable.

Data encryption is becoming more commonplace with some vendors and industries (for example, the payment card industry). Use data encryption only when it is required or for very high-value sensitive data. In some cases, encrypting the network channel or using SQL Server permissions is a better choice because of the complexity involved in managing keys and invoking encryption/decryption routines.

Because unencrypted data must be stored in memory buffers before being transmitted to clients, it is impossible to keep data away from an administrator who has the ability to debug the process or to patch the server. Memory dumps can also be a source of unintended data leakage. If symmetric keys are protected by asymmetric keys and the asymmetric keys are encrypted by using the database master key, a database administrator could impersonate a user of encrypted data and access the data through the keys. If protection from the database administrator is preferred, encryption keys must be secured by passwords, rather than by the database master key. To guard against data loss, encryption keys that are secured by passwords must have an associated disaster recovery policy (offsite storage, for example) in case of key loss. You can also require users to specify the database master key by dropping encryption of the database master key by the instance master key. Remember to back up the database in order to back up the symmetric keys, because there are no specific DDL statements to back up symmetric and asymmetric keys, just as there are specific DDL statements to back up certificates, the database master key, and the service master key.

### **Best practices for data encryption**

- Encrypt high-value and sensitive data.
- Use symmetric keys to encrypt data, and asymmetric keys or certificates to protect the symmetric keys.

- Password-protect keys and remove master key encryption for the most secure configuration.
- Always back up the service master key, database master keys, and certificates by using the key-specific DDL statements.
- Always back up your database to back up your symmetric and asymmetric keys.

### *Auditing*

SQL Server 2005 supports login auditing, trigger-based auditing, and event auditing by using a built-in trace facility. Password policy compliance is automatically enforceable through policy in SQL Server 2005 for both Windows logins and SQL logins. Login auditing is available by using an instance-level configuration parameter. Auditing failed logins is the default, but you can specify to audit all logins. Although auditing all logins increases overhead, you may be able to deduce patterns of multiple failed logins followed by a successful login, and use this information to detect a possible login security breach. Auditing is provided on a wide variety of events including Add Database User, Add Login, DBCC events, Change Password, GDR events (Grant/Deny/Revoke events), and Server Principal Impersonation events. SQL Server 2005 SP2 also supports login triggers.

SQL Server 2005 introduces auditing based on DDL triggers and event notifications. You can use DDL triggers not only to record the occurrence of DDL, but also to roll back DDL statements as part of the trigger processing. Because a DDL trigger executes synchronously (the DDL does not complete until the trigger is finished), DDL triggers can potentially slow down DDL, depending on the content and volume of the code. Event notifications can be used to record DDL usage information asynchronously. An event notification is a database object that uses Service Broker to send messages to the destination (Service Broker-based) service of your choosing. DDL cannot be rolled back by using event notifications.

Because the surface area of SQL Server 2005 is larger than previous versions, more auditing events are available in SQL Server 2005 than in previous versions. To audit security events, use event-based auditing, specifically the events in the [security audit event category](#) (listed in SQL Server Books Online). Event-based auditing can be trace-based, or event notifications-based. Trace-based event auditing is easier to configure, but may result in a large event logs, if many events are traced. On the other hand, event notifications send queued messages to Service Broker queues that are in-database objects. Trace-based event auditing cannot trace all events; some events, such as SQL:StmtComplete events, are not available when using event notifications.

There is a WMI provider for events that can be used in conjunction with SQL Server Agent alerts. This mechanism provides immediate notification through the Alert system that a specific event has occurred. To use the WMI provider, select a WMI-based alert and provide a WQL query that produces the event that you want to cause the alert. WQL queries use the same syntax for naming as does event notifications. An example of a WQL query that looks for database principal impersonation changes would be:

```
SELECT * FROM AUDIT_DATABASE_PRINCIPAL_IMPERSONATION_EVENT
```

SQL Server can be configured to support auditing that is compliant with C2 certification under the Trusted Database Interpretation (TDI) of the Trusted Computer System Evaluation Criteria (TCSEC) of the United States National Security Agency. This is known as *C2 auditing*. C2 auditing is configured on an instance level by using the **C2 audit mode** configuration option in **sp\_configure**.

When C2 auditing is enabled, data is saved in a log file in the Data subdirectory in the directory in which SQL Server is installed. The initial log file size for C2 auditing is 200 megabytes. When this file is full, another 200 megabytes is allocated. If the volume on which the log file is stored runs out of space, SQL Server shuts down until sufficient space is available or until the system is manually started without auditing. Ensure that there is sufficient space available before enabling C2 auditing and put a procedure in place for archiving the log files.

SQL Server 2005 SP2 allows configuring an option that provides three elements required for Common Criteria compliance. The Common Criteria represents the outcome of efforts to develop criteria for evaluation of IT security that are widely useful within the international community. It stems from a number of source criteria: the existing European, US, and Canadian criteria (ITSEC, TCSEC, and CTCPEC respectively). The Common Criteria resolves the conceptual and technical differences between the source criteria. The three Common Criteria elements that can be configured by using an instance configuration option are:

- Residual Information Protection, which overwrites memory with a known bit pattern before it is reallocated to a new resource.
- The ability to view login statistics.
- A column-level GRANT does not override table-level DENY.

You can configure an instance to provide these three elements for Common Criteria compliance by setting the configuration option **common criteria compliance enabled** as shown in the following code.

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'common criteria compliance enabled', 1;
GO
RECONFIGURE;
GO
```

In addition to enabling the Common Criteria options in a SQL Server instance, you can use login triggers in SQL Server 2005 SP2 to limit logins based upon time of day or based on an excessive number of existing connections. The ability to limit logins based on these criteria is required for Common Criteria compliance.

### Best practices for auditing

- Auditing is scenario-specific. Balance the need for auditing with the overhead of generating additional data.
- Audit successful logins in addition to unsuccessful logins if you store highly sensitive data.
- Audit DDL and specific server events by using trace events or event notifications.
- DML must be audited by using trace events.
- Use WMI to be alerted of emergency events.
- Enable C2 auditing or Common Criteria compliance only if required.

### *Microsoft Baseline Security Analyzer and SQL Server Best Practices Analyzer*

Microsoft Baseline Security Analyzer (MBSA) is a utility that scans for common insecurities in a SQL Server configuration. Run MBSA on a regularly scheduled basis, either locally or across the network. MBSA scans for Windows operating system, security principal, network, and file system insecurities and tests for SQL Server 2000 and MSDE-specific insecurities, but does not incorporate SQL Server 2005-specific checks yet. It will check to see if SQL Server 2005 is patched to the latest Service Pack version.

SQL Server 2005 Best Practices Analyzer 2.0 CTP has been released in conjunction with Service Pack 2. You can download it from the Microsoft Download Center, [SQL Server 2005 Best Practices Analyzer \(February 2007 CTP\)](#) page. SQL Server 2005 Best Practices Analyzer (BPA) gathers data from Microsoft Windows and SQL Server configuration settings. Best Practices Analyzer uses a predefined list of SQL Server 2005 recommendations and best practices to determine if there are potential security issues in the database environment.

#### **Best practice analysis utilities recommendations**

- Run BPA against SQL Server 2005.
- Regularly run MBSA 2.0 to ensure latest SQL Server 2005 patch level
- Regularly run MBSA 2.0 for SQL Server 2000 instances

### *Patching*

The best way to ensure the security of the server software and to ensure the security of SQL Server 2005 is to install security hotfixes and service packs as soon as possible. Use manual updates on an operating system basis by using Windows Update or Microsoft Update. You can enable automatic updates using Windows Update or Microsoft Update as well, but updates should be tested before they are applied to production systems.

SQL Server 2005 incorporates SQL Server hotfixes and service packs into Windows Update. All hotfixes should be installed immediately and service packs should be tested and installed as soon as possible. This requirement cannot be emphasized enough. For suggestions on minimizing downtime when installing hotfixes and service packs, see [Preventing Reboots, Installing Multiple Updates, and More](#) on Microsoft TechNet.

#### **Best practices for patching SQL Server**

- Always stay as current as possible.
- Enable automatic updates whenever feasible but test them before applying to production systems.

### *Conclusion*

Security is a crucial part of any mission-critical application. To implement security for SQL Server 2005 in a way that is not prone to mistakes, security setup must be relatively easy to implement. The "correct" security configuration should be the default configuration. This paper describes how it is a straightforward task to start from the SQL Server 2005 security defaults and create a secure database configuration according to the Trustworthy Computing Initiative guidelines.

**For more information:**

<http://www.microsoft.com/technet/prodtechnol/sql/themes/security.msp>

Did this paper help you? Please give us your feedback. On a scale of 1 (poor) to 5 (excellent), [how would you rate this paper?](#)