

SQL Server 2012 Security Best Practices - Operational and Administrative Tasks

SQL Server White Paper

**Author:** Bob Beauchemin, SQLskills

**Technical Reviewers:** Darmadi Komo, Jack Richins, Devendra TiwariDarmadi Komo

**Published:** January 2012

**Applies to:** SQL Server 2012

**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 2012.

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.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, <plus, in alphabetical order, all Microsoft trademarks used in your white paper> are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Contents

[Introduction 3](#_Toc310427896)

[Compliance 4](#_Toc310427897)

[Surface Area Reduction 4](#_Toc310427898)

[Policy-Based Management 6](#_Toc310427899)

[Service Account Selection and Management 7](#_Toc310427900)

[SQL Server Best Practices Analyzer and other analysis utilities 10](#_Toc310427901)

[Patching and Automatic Windows Update 10](#_Toc310427902)

[Industry Compliance 11](#_Toc310427903)

[Encryption 12](#_Toc310427904)

[Data and Database Encryption 12](#_Toc310427905)

[SSL Encryption 16](#_Toc310427906)

[Access Control 17](#_Toc310427907)

[Administrator Privileges 17](#_Toc310427908)

[Server Roles 17](#_Toc310427909)

[Database Ownership and Trust 17](#_Toc310427910)

[Lockdown of System Stored Procedures 18](#_Toc310427911)

[Schemas 20](#_Toc310427912)

[Authorization 21](#_Toc310427913)

[Catalog Security 23](#_Toc310427914)

[Execution Context 24](#_Toc310427915)

[Remote Data Source Execution 26](#_Toc310427916)

[Authentication 27](#_Toc310427917)

[Authentication Modes and Logins 27](#_Toc310427918)

[Using Contained Databases 28](#_Toc310427919)

[Password Policy 28](#_Toc310427920)

[Network Security 29](#_Toc310427921)

[Auditing 32](#_Toc310427922)

[Conclusion 35](#_Toc310427923)

## Introduction

This white paper covers some of the operational and administrative tasks associated with Microsoft® SQL Server™ 2012 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 2012 Books Online. Features and options that are new or defaults that are changed for SQL Server 2012 are identified. The target audience for this whitepaper is the Operational DBA, that is, the administrator responsible for SQL Server setup and day-to-day operations, and provides prescriptive guidance on a task-by-task basis. Coding examples for operational tasks use Transact-SQL, so understanding Transact-SQL is required for you to get the most out of this paper.

## Compliance

### Surface Area Reduction

SQL Server 2012 installation minimizes the "attack surface" because by default, optional features are not installed. During installation the administrator can choose to install:

* Database Engine
  + SQL Server Replication
  + Full-text and Semantic Extractions for Search
  + Data Quality Services
* Analysis Services Engine
* Reporting Services – Native and/or SharePoint
* Integration Services
* Master Data Services
* Distributed Replay Server and Client
* Various Tools and Clients
* Documentation Components

It is a best practice to review which product features you actually need and install only those features. Later, install additional features only as needed. SQL Server 2012 does not install any samples or sample databases; the official SQL Server samples are now available on the CodePlex website. Each item of official sample code has undergone a review to ensure that the code follows best practices for SQL Server 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 2012 can be overwhelming. One way to make a system more secure is to limit the number of optional engine 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 2012, 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 2012 contains a Policy-Based Management feature that makes it possible to define, deploy, and validate policy, and thereby enforce best practices. This feature subsumes and extends the SQL Server Surface Area Configuration tool that was introduced in SQL Server 2005. Although the Surface Area Configuration tool does not ship with SQL Server 2008 and above, Microsoft provides some predefined policies that correspond to the settings that were managed with the Surface Area Configuration tool. These policies are available as XML files and include:

* Surface Area Configuration for Database Engine 2005 and 2000 Features
* Surface Area Configuration for Database Engine 2008 Features
* Surface Area Configuration for Service Broker Endpoints
* Surface Area Configuration for SOAP Endpoints (note that SOAP endpoints are no longer available in SQL Server 2012)
* Surface Area Configuration for Analysis Services Features
* Surface Area Configuration for Reporting Services 2005 Features
* Surface Area Configuration for Reporting Services 2008 Features

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 Policy-Based Management combines this functionality into a single feature for ease-of-use. Policy-Based Management is usually configured through the graphic user interface, but can also be configured through system stored procedures or by using PowerShell scripts.

SQL Server Policy Based Management’s Surface Area Configuration policies divide configuration into two subsets: endpoints and features. The database engine features included in the predefined policy are:

* CLR Integration
* Remote use of a dedicated administrator connection
* OLE Automation system procedures
* System procedures for Database Mail and SQL Mail (note: SQL Mail is no longer supported in SQL Server 2012)
* Ad hoc remote queries (the OPENROWSET and OPENDATASOURCE functions)
* **xp\_cmdshell** availability
* SQL Server Web Assistant (this was removed in SQL Server 2008)

You can use the SQL Server Configuration Manager tool to view the installed components of SQL Server and the client network interfaces. The startup type for each service (Automatic, Manual, or Disabled) and the server network interfaces that are available can be configured on a per-instance basis.

As of SQL Server 2005, SQL Server Browser functionality was 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 (note: this service was removed in SQL Server 2012)
* SQL Server Agent
* SQL Server Full-text Filter Daemon Launcher
* SQL Server Browser
* SQL Server Distributed Replay Client
* SQL Server Distributed Replay Controller
* 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 2012 and named instances that use static TCP/IP ports if they use the well-known SQL Server port or specific port number as part of the connection string. 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.

**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 Policy-Based Management to standardize this policy.
* Develop a policy for the usage of optional features. Use SQL Server Policy-Based Management 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.
* Configure only those server network interfaces that you will actually use.

### Policy-Based Management

Policy-Based Management can not only be used to manage and configure the Surface Area, but can also be used to detect out-of-compliance conditions, although there is no strong guarantee that it can stop any behavior that would put you out of compliance with the policy. In addition to the Surface Area Configuration policies mentioned previously, SQL Server 2012 includes a set of security best practices policies. These policies include:

* Asymmetric Key Encryption Algorithm
* CmdExec Rights Secured
* Guest Permissions
* Public Not Granted Server Permissions
* SQL Server Login Mode
* SQL Server Password Expiration
* SQL Server Password Policy
* Symmetric Key Encryption for User Databases
* Symmetric Key for master Database
* Symmetric Key for System Databases
* Trustworthy Database

You can implement these policies as-is, or customize them to enforce and insure compliance with your shop standards. Some sample policies to assist in monitoring compliance are provided as part of the SQL Server 2008 Compliance Guide and also in the whitepaper “Deploying SQL Server 2008 Based on Payment Card Industry Data Security Standards (PCI DSS)” by John Bastow.

SQL Server Policy-Based Management allows you to export and import policies using XML files and includes a PowerShell cmdlet that accomplishes the policy evaluation and can be used to script policy evaluation on a custom schedule.

SQL Server 2008 introduced a feature called Central Management Server to make it easier to manage groups of SQL Server instances. Using a Central Management Server you can evaluate a set of policies against a set of SQL Server instances. This enables you to standardize the configuration of a group of SQL Server 2012 instances. Note that, using a Central Management Server, you can evaluate policies against SQL Server 2000 and above instances as well as SQL Server 2012.

As an adjunct and extension to the Policy-Based Management feature, the [Enterprise Policy Management Framework](http://epmframework.codeplex.com/) CodePlex project allows management, history storage, and reporting on a set of SQL Server instances using a Central Management Server, PowerShell scripts, a history database, and SQL Server Reporting Services.

**Best practices for policy-based management**

* Develop a policy for network connectivity, usage of optional features, and the implementation of SQL Server security best practices. Use SQL Server Policy-Based Management to standardize this policy.
* Use Central Management Servers to standardize and enforce security policies across sets of servers in the enterprise.
* Use Enterprise Policy Management Framework to consolidate history and reporting of sets of enterprise-wide policies.

### Service Account Selection and Management

SQL Server 2012 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 Management. When you select a Windows account to be a SQL Server service account, you have a choice of:

* Virtual Service account (new in SQL Server 2012)
* Managed Service account (new in SQL Server 2012)
* 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 four account types in the list above (and Network Service account when running on Windows Server 2008 and above operating systems) have both of these properties. Making the SQL Server service account an administrator, at either a server level or a domain level, or using Local System, bestows too many unneeded privileges.

Windows Server 2008 R2 and Windows 7 operating systems introduced two new types of service accounts: Virtual accounts and managed service accounts. SQL Server 2012 is the first version of SQL Server to permit these accounts to be used as service accounts. These are introduced to ensure isolation of windows services and to make provisioning, password management, and SPN configuration easier to manage.

A managed service account is a special type of domain account that can be assigned to a single computer and used to manage a service. It must be provisioned by the domain administrator prior to being used. This type of account cannot be used to log in to a computer and provides automatic SPN and password management, once provisioned.

Virtual accounts are managed local accounts that is automatically provisioned and managed. In SQL Server 2012, they are the default service account specified during setup. It can access the network. A virtual service account has a well-known name in the form of NT SERVICE\<SERVICENAME> and can access the network using the credentials <domain\_name>\<computer\_name>$.

Windows Server 2008 and Windows Vista introduced per-service security identifiers (known as per-service SIDs). A per-service SID creates, in essence, an identity for each service which enables access control using the existing Windows access control model.  Services can now apply explicit access control lists (ACL's) to resources that are private to the service - preventing other services as well as the user from accessing that resource.

On a Windows 2008 R2 or Windows 7 operating system, managed service accounts and virtual service accounts are the best choices. With the introduction of Service SIDs in Windows Server 2008, Network Service is a good choice, and alleviates the need to change service passwords. Using a local user or domain user that is not a Windows administrator is also a good 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 managed service 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) and does not need to access domain resources, a virtual account is preferred.

Creating the user account that will be used as a SQL Server service account is easier in SQL Server 2005 or later than in previous versions. When SQL Server 2005 or later 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.

In SQL Server 2012, only SQL Server Analysis Services and the SQL Browser Service ever result in the creation of Windows groups. Because Service SIDs are always used, the Service SID is added to the group for these services, rather than the service account specified at installation time.

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 SQLServerMSASUser$computer\_name$instance\_name group for the “instance\_name” instance of SQL Server Analysis Services on server "computer\_name."

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, although this happens automatically when SQL Server starts, thanks to a redundant copy of the service master key protected by DPAPI (machine key), even if the service account is changed from the Windows SCM. 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 and later, it is easier to abide by password expiration policies because changing the password of the service account does not require restarting SQL Server.

The SQL Server Agent service account requires **sysadmin** privilege in the SQL Server instance that it is associated with. In SQL Server 2005 and above, 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)
* PowerShell

**Best practices for SQL Server service accounts**

* On Windows 2008 R2 or Windows 7 operating systems use managed service accounts and virtual server accounts
* On Windows 2008 SP2 or Windows Vista SP2 operating systems, use the Network Service account or a specific user account or domain account rather than a shared account for SQL Server services.
* Always use SQL Server Configuration Manager to change service accounts.
* If you use a user or domain account, 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 a 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.

### SQL Server Best Practices Analyzer and other analysis utilities

SQL Server 2008 R2 Best Practices Analyzer has been released and you can download it from the Microsoft Download Center, [SQL Server 2008 R2 Best Practices Analyzer](http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=0fd439d7-4bff-4df7-a52f-9a1be8725591) page. SQL Server 2008 R2 Best Practices Analyzer (BPA) gathers data from Microsoft Windows and SQL Server configuration settings. Best Practices Analyzer uses a predefined list of SQL Server 2008/2008 R2 recommendations and best practices to determine if there are potential security issues in the database environment.

You can also use Microsoft Security Compliance Manager to manage security in general on machines running SQL Server. Part of Trustworthy Computing, this tool provides centralized security baseline management features, a baseline portfolio, customization capabilities, and security baseline export flexibility to efficiently manage the security and compliance process.

Finally, Knowledge Base article 309422 discusses “How to choose antivirus software to run on computers that are running SQL Server”. Be sure and consult this article before setting up any antivirus software on a SQL Server machine.

**Best practice analysis utilities recommendations**

* Run SQL Server Best Practices Analyzer against SQL Server 2008/2008 R2.
* Use Microsoft Security Compliance Manager to provide centralized security baseline management.
* Consider running antivirus software on the SQL Server machine if appropriate.

### Patching and Automatic Windows Update

The best way to ensure the security of the server software and to ensure the security of SQL Server 2012 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 and above incorporate 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](http://www.microsoft.com/technet/technetmag/issues/2006/12/SQLQA/default.aspx) 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.

### Industry Compliance

SQL Server 2008 R2 has achieved Common Criteria Compliance, allowed C2 auditing, helping administrators achieve industry compliance such as PCI, HIPAA, and FIPS 140-2. SQL Server 2012 Common Criteria Compliance certificate is in progress. For more information in general refer to the [Common Criteria Portal](http://www.commoncriteriaportal.org/). For more information about C2 and Common Criteria Auditing, refer to the Auditing section of this whitepaper. For more information about complying with specific industry standards, refer to whitepapers such as [SQL Server 2008 in FIPS 140-2-compliance mode](http://support.microsoft.com/kb/955720), [PCI Compliance with SQL Server 2008](http://www.parentebeard.com/Uploads/Files/Deploying_SQL_Server_2008_Based_on_PCI_DSS.PDF) and the associated [webcast](http://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?culture=en-US&EventID=1032404174&CountryCode=US), [HIPAA Compliance with SQL Server 2008](http://www.jeffersonwells.com/mssql2008hipaa) and the associated [webcast](http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032441700&Culture=en-US).

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. Both SQL Server 2005 SP2 and SQL Server 2008 have been certified to conform to the Common Criteria Specification. At the time of this writing, the following Common Criteria assurance levels were achieved by the specified versions of SQL Server:

* SQL Server 2008 Enterprise Edition build 10.0.4000.0 (SP2) - EAL4+, ALC\_FLR.2
* SQL Server 2008 Enterprise Edition build 10.0.1600.0 - EAL1+, ASE\_OBJ.2, ASE\_REQ.2, ASE\_SPD.1
* SQL Server 2005 Enterprise Edition build 9.00.2047.00 (SP2) – EAL4+
* SQL Server 2005 Enterprise Edition build 9.00.2047.00 (SP1) – EAL1

The Common Criteria certification includes a required audit configuration (supplied as a trace setup DDL file), a product integrity check procedure, and a Guidance Addendum / Installation/ Startup document.

SQL Server 2005 SP2 and above allows configuring an option that provides three elements required for Common Criteria compliance. These elements can be configured by using an instance configuration option:

* 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 and above 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.

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. For more information about C2 auditing reference the Auditing section of this whitepaper

**Best practices for compliance with industry standards**

* Use the guides provided at the SQL Server Compliance portal as adjuncts to achieving compliance with specific standards
* Enable C2 auditing or Common Criteria compliance only if required.

## Encryption

### Data and Database Encryption

SQL Server 2012 has built-in data encryption, both at a cell level and encryption of an entire database. Data encryption at a cell level and is accomplished by means of built-in system procedures. Database-level data encryption (known as transparent data encryption or TDE) is accomplished by using DDL statements. TDE is an Enterprise version-only feature introduced in SQL Server 2008; cell level encryption has been around since SQL Server 2005.

Encrypting data requires secure encryption keys and key management. A key management hierarchy is built into SQL Server. 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. In SQL Server 2012, the service and database master keys use the AES\_256 encryption algorithm; previous versions used Triple DES encryption.

SQL Server 2005 and above 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. Integrity check 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 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 revocation 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. In SQL Server 2012, when creating certificates from external sources, the maximum length of private keys imported from an external source is expanded from 3,456 to 4,096 bits. SQL Server 2012 also allowed the CREATE CERTIFICATE DDL statement to specify a FROM BINARY option to allow specifying the binary description of an ASN encoded certificate. The new functions CERTENCODED and CERTPRIVATEKEY can be used to extract a binary description of an existing certificate, making certificates easier to deploy using DDL.

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 and RC4\_128 are supported only for backward compatibility when the database compatibility level is 90 or 100, and should not be used. User-defined algorithms are not supported. The key algorithm and key length choice should be predicated on the sensitivity of the data. The HASHBYTES function in SQL Server 2012 supports the SHA2\_256, and SHA2\_512 algorithms in addition to hashing algorithms supported in previous SQL Server versions.

The **master** database in SQL Server contains some built-in certificates with names that begin with “##MS\_”. These certificates are used for system functionality and should not be deleted.

As an alternative for some parts of the key management built into SQL Server, SQL Server 2008 introduced support for Extensible Key Management. Using Extensible Key Management (EKM) means that keys can be managed by an external source, such as a hardware security module. The external source is referred to in SQL Server as a cryptographic provider. TDE supports asymmetric keys that are provisioned by EKM. No other form of asymmetric key is supported by TDE and database certificates cannot currently be provisioned through EKM. EKM is supported for cell-level encryption through symmetric and asymmetric keys. It is highly recommended that you use EKM with both database-level and cell-level encryption for more comprehensive key management and hardware-based cryptography. Extensible Key Management is an Enterprise-version only feature.

SQL Server can encrypt 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](http://msdn.microsoft.com/en-us/library/ms345262(v=SQL.110).aspx) 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.

SQL Server 2008 introduced database-level encryption as an adjunct and alternative to cell-level encryption. In addition, recent versions of the Microsoft Windows operating system (Windows Server 2008 and beyond and some versions of Windows Vista and Windows 7) provide disk volume encryption using BitLocker. Both TDE and BitLocker encrypt data as it in written to disk and decrypt data as it is read from disk. Either one can be added to the disk storage media without any change to the database. Access to the encrypted data in SQL Server is controlled through normal SQL Server permissions.

Transparent Data Encryption encrypts data files, log files, tempdb, and database backups, but does not encrypt data stored using SQL Server’s FILESTREAM storage feature. BitLocker encrypts the entire volume, which would include FILESTREAM storage and non-SQL Server database files. As a general rule, it probably makes more sense to use BitLocker on portable computers (even those without SQL Server) as it protects the entire volume. If laptop data is compromised, presumably this happens due to physical access to the media and BitLocker protects against this. On a server, however, it probably makes more sense to use SQL Server TDE. If a server storage volume is compromised, it is typically through network access and BitLocker does not protect against this because when using TDE, the encryption is bound to the database files, not to the disk, in such a way that even if the database files are copied to another drive/tape, they will continue being encrypted. BitLocker is volume-bound, so copying to another disk would effectively copy a cleartext file.

Note that you can also perform encrypted backups on all SQL Server databases using Data Protection Manager (DPM). Part of System Center, DPM provides seamless and continuous protection of multiple SQL Server backups to integrated disk, tape, and cloud. For more information, download the SQL Server-specific whitepapers from the [Microsoft System Center website](http://www.microsoft.com/systemcenter/en/us/data-protection-manager/dpm-protect-sql.aspx).

Enabling TDE on a database requires two keys: a database encryption key that uses symmetric key encryption to affect the encryption of the database and a certificate in the master database that protects the database encryption key. Because the certificate in the master database is needed to restore the database, care should be taken to back this certificate up.

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, Extensible Key Management is the first choice, followed by encryption keys secured by passwords if EKM is not available, 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.

For more information about SQL Server encryption, see the whitepaper [Database Encryption in SQL Server 2008](http://download.microsoft.com/download/8/b/2/8b22991f-3f2f-4cea-b2ba-55c190841145/TDEandEFSBitLocker.docx).

**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.
* Do not delete pre-provisioned system certificates in the **master** database
* 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.
* TDE is recommended for encrypting existing applications or for performance sensitive applications.
* Cell-level encryption can be used for defense in depth both for a database encrypted by TDE and for limited access control through the use of passwords.
* Use EKM with both database-level and cell-level encryption for more comprehensive key management and hardware-based cryptography.

### SSL Encryption

SQL Server 2012 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 Microsoft OLE DB, ODBC, JDBC 1.2, and ADO.NET clients all support encrypted sessions. 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 2012 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.

Bear in mind that the TDS protocol version that SQL Server uses is negotiable. Using an older version of TDS (e.g. older versions of FreeTDS) to connect to SQL Server may result in the encrypted channel for login not being used. Therefore, it is a best practice to always use modern drivers/providers with SQL Server.

**Best practices for SSL channel encryption**

* If you must support SQL logins, install an SSL certificate from a trusted certificate authority rather than using SQL Server 2008 self-signed certificates.
* Use "allow only encrypted connections" only if needed for end-to-end encryption of sensitive sessions.
* Do not use older TDS protocol versions; this ensures that SQL Login information is always encrypted.

## Access Control

### Administrator Privileges

SQL Server 2008 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 was introduced in SQL Server 2005. Ensure that your auditing procedures include any login with CONTROL SERVER permission.

Beginning in SQL Server 2008, SQL Server does not automatically grant the server's Administrators group (BUILTIN\administrators) the **sysadmin** server role. When installing SQL Server 2012, you are required to add at least one Windows login to the **sysadmin** role. This is true for SQL Server Analysis Services as well as the database engine.

When running SQL Server Express, setup incorporates the specification of a specific principal to act as administrator. SQL Server Express setup 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 (ADDCURRENTUSERASSQLADMIN). For more detailed information, see [How to: Install SQL Server 2012 from the Command Prompt](http://msdn.microsoft.com/en-us/library/ms144259.aspx) in SQL Server 2012 Books Online.

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.
* Have multiple distinct administrators if more than one is needed.
* Avoid dependency on the builtin\administrators Windows group.

### User-Defined Server Roles

SQL Server 2012 introduces user-defined server roles, which can function in an analogous way to user-defined database roles, but with server-level securables. This allows you to collect sets of permissions and assign them a role, rather than granting them to individual users. Any server-level permission, such as CREATE ANY DATABASE, ALTER ANY DATABASE, CONNECT SQL, or SHUTDOWN can be granted to a server role. Using user-defined server roles rather than granting access to individuals, should be considered a best practice.

### 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 *ISP 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 (ISP) 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 ISP 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 OFF

Enabling the TRUSTWORTHY property may open an Elevation of Privilege path from **DBO** to **sysadmin** (depending on the permissions of the DBO for that database), therefore it is highly recommended to not enable the TRUSTWORTHY property on any user-defined database that is owned by a **sysadmin**.

**Best practices for database ownership and trust**

* Use user-defined server roles as an alternative to assigning server-level privileges to individual users.
* If you are an ISP, have distinct owners for databases; not all databases should be owned by **sa**.
* 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.

### 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.

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 and above, 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 Policy-Based Management. These include:

* **xp\_cmdshell** - executes a command in the underlying operating system
* Database Mail procedures
* SQL Mail procedures (SQL Mail is removed in SQL Server 2012)
* COM component procedures (e.g. **sp\_OACreate**)

Enable these procedures only if necessary.

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 Policy-Based Management 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 modify the default permissions on system objects.
* Do not DENY all users/administrators access to the extended procedures.

### Schemas

SQL Server 2005 introduced 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 2012, the table would have to be part of a schema. If **payroll\_app** is the name of the SQL Server 2012 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 2012, 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**.

In SQL Server 2012, users that are mapped to Windows groups cannot have default schemas. If a Windows user is a memory of more than one Windows group, the default schema for that user is the default schema for the Windows group with the highest principal\_id.

For more information on using schemas reference the whitepaper [SQL Server 2008 Database Object Schemas Best Practice](http://msdn.microsoft.com/en-us/library/dd283095.aspx).

**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 or use a user without a login as a schema owner.
* Not all schemas should be owned by **dbo**.
* Minimize the number of owners for each schema.
* For users mapped to Windows groups, try and limit each Windows user to one Windows group that has database access.
* Use two-part names for database object creation and access.

### 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. However, in rare cases (e.g. permissions granted to use DBCC subcommands), role permissions are still used.

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 and above contain 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 only 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 except in the MSDB database. In order for some SQL Server features to work, the guest user must be enabled in MSDB.

SQL Server 2005 introduced 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](#_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

SQL Server does not yet implement row or cell-level security as part of the database engine. However, the SQL Server Label Security Toolkit is available as a CodePlex project. The Label Security Toolkit provides tools and techniques for using Microsoft® SQL Server (versions 2005 through 2008 R2) to implement row-level security (RLS) and cell-level security (CLS) based on security labels. This toolkit is meant to be used as an adjunct and a complement to the SQL Server whitepaper, [Implementing Row- and Cell-Level Security in Classified Databases](http://download.microsoft.com/download/8/8/0/880F282A-AA4E-4351-83C0-DFFA3B56A19E/SQL_Server_RLS-CLS_White_paper.docx).

For more information about database object authorization strategies, reference the technical article [SQL Server 2008 Separation of Duties for Application Developer](http://msdn.microsoft.com/en-us/library/cc974525.aspx).

**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 in any database except MSDB.
* 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 and later, 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. As of 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 and above 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. Note that VIEW DEFINITION will also allow the caller to see the definition of objects such as stored procedures.

**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 from SQL Server 2000.

### 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 and above, 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 object creator (the user who executes the DDL statement that creates the object), 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 and above offer 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

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 CERTIFCATE 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 = 'HacdeNj162kqT') -- must match CREATE password

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='somelogin' 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. You can also get the name of the caller by including an “EXECUTE AS CALLER” block in the stored procedure or module and fetching SUSER\_NAME() there.

**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.

### 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 an above.

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 the whitepaper [How to Implement Kerberos Constrained Delegation with SQL Server 2008](http://msdn.microsoft.com/en-us/library/ee191523(SQL.100).aspx).

**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.

## Authentication

### Authentication Modes and Logins

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 Kerberos or NTLM authentication credentials. SQL Server 2008 can use Kerberos authentication with all protocols; previous versions only used Kerberos with the TCP/IP protocol. 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. Other uses for SQL logins are with cross-platform client-server applications in which the non-Windows clients do not possess Windows logins and applications that require logins from untrusted domains. Although using SQL logins is discouraged, there are security improvements for SQL logins in SQL Server 2005 and later. 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 and above use 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 and above also provide 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\_revokelogin**, 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.

SQL Server contains some pre-defined logins such as NT AUTHORITY\SYSTEM and ##MS\_PolicyEventProcessingLogin##. These logins are used for SQL Server built-in functionality and should not be deleted.

Logins can be based on Windows Groups in addition to being based on Windows Users. Using Windows Logins instead of Windows Group provides the ability to identify individual Windows Users for tracking purposes. Also, because a SQL Server login is tied to a default database and default language, if Windows Groups are used for logins and a Windows Users is a member of multiple Windows Groups with SQL Server logins, the default database and default language is determined by using the Windows Group with the highest principal\_id. Therefore, attempt to keep Windows Group logins relatively consistent with respect to default database and language, if users can belong to multiple groups.

SQL Server 2005 SP2 introduced login triggers. Logon Triggers provide extra control of access to the database by allowing custom actions during the login process.

**Best practices for authentication mode and logins**

* Always use Windows Authentication mode if possible.
* Use Mixed Mode Authentication only for legacy applications, non-Windows users, and users from untrusted domains.
* Use the standard login DDL statements instead of the compatibility system procedures.
* It the **sa** account is not going to be used, you should disable it. 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 knows user or group.
* Rename the **sa** account to a different account name to prevent attacks on the **sa** account by name.
* Do not delete internal built-in logins
* Use Windows Logins rather than Windows Group to control access to SQL Server and use care when using Windows Group logins to prevent group overlap for a particular user.
* Use login triggers for more granular control of the login process.

### 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 or Group Policy objects. 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 and above, SQL logins can also go by the login policies of the underlying 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 2008, 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 the Encryption section of this whitepaper.

**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, use password policies.
* Outfit your applications with a mechanism to change SQL login passwords.
* Set MUST\_CHANGE for new logins.

### Contained Databases and Authentication

SQL Server 2012 introduced the concept of contained databases. A contained database is a database that includes all of the settings and metadata needed for its operation with dependencies on the instance. From a security perspective, a contained database makes it easier to have user accounts that are limited to a single database. SQL Server 2012 supports partially contained databases; it does not yet support full containment. Although contained databases allow more control by the “application administrator” they do have security repercussions.

Contained databases are interesting for a security point of view in that they allow defining users with authentication privileges, i.e. users who can log directly into a contained database without a corresponding login. Contained databases support two types of users: Windows users and groups that can directly connect to the database and do not need logins, and users with a password where the password is authenticated by the database, not the instance. This is not permitted by default; the instance administrator must specifically allow it by setting the “contained database authentication” configuration option on. To disallow connections to specific contained databases but allow access to other contained databases, a login trigger can be used.

Allowing direct connection of users to a database changes the effective threat-level of some existing permissions. For example, the permission ALTER ANY USER in a contained database gives permission to add user-based access to an instance through a contained database. Logins with ALTER ANY DATABASE or CONTROL DATABASE permission could also set containment on a database and add instance access through users. In addition, contained database users can connect to other database in the instance if the guest account is enabled.

Although you can use the system stored procedure sp\_migrate\_user\_to\_contained to migrate a database user mapped to a SQL login to a contained database user-with-password, contained databases’ user-with-password does not use password history and expiration policies as SQL logins do, although password complexity checks are still used. Therefore, attaching a contained database could allow weak passwords. In addition, the password hashes for these passwords are stored in the database, not in master. Anyone with access to the database file could perform a dictionary attack on a separate, unaudited instance.

There exists the possibility of conflicts, if a login and contained database user have the same name. The rule for resolving this conflict is that if an initial catalog is specified in the connection string and it’s the contained database, access is checking against the user based principal, not the login. To ameliorate this possibility, do not create conflicting names or specify a contained database as an initial catalog in a connection string. In addition, members of the sysadmin role should not use initial catalog in a connection string.

**Best practices for contained databases**

* Use the default (off) setting for contained database authentication and only turn this setting on if it is required.
* Protect backups of contained databases using passwords.
* Audit the capabilities of users and modules in contained databases.
* Audit logins that have the ability to set containment, if contained database authentication is allowed.
* Disable the guest account on databases that share an instance with contained databases.
* Take care to avoid login/user-with-login naming conflicts
* Avoid connection strings with initial catalog if contained database authentication is permitted.

## Network Security

A standard network protocol is required to connect to the SQL Server database. There are no internal connections that bypass the network. As part of SQL Server 2012 installation, a warning message will occur if Windows Firewall is not enabled on the server machine. It is a general network security best practice to enable Windows Firewall and restrict network protocols and ports to the minimum necessary for SQL Server operation.

SQL Server 2005 introduced 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
* Dedicated administrator connection

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

* Service Broker
* Database mirroring
* HTTP Web Services (these endpoints are removed in SQL Server 2012)

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 )

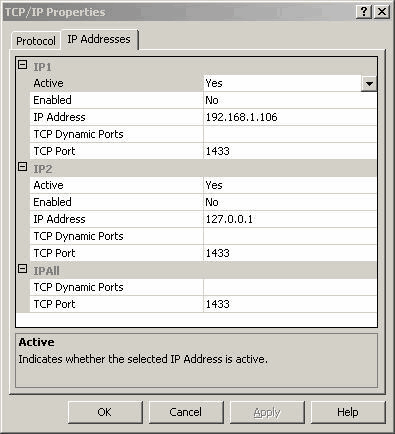
In keeping with the general policy of "off by default, enable only when needed," no Service Broker, HTTP, or database mirroring endpoints are created when SQL Server 2012 is installed. In addition, in SQL Server 2012 Express Edition, SQL Server 2012 Developer Edition, and SQL Server 2012 Evaluation Edition, 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) that was added in 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.

Additional ports are required to enable NETBIOS protocol (UDP/137, UDP/138, and TCP/139) and SMB (Server Message Block) protocol (TCP/139 and TCP/445). These should be disabled by default on a machine running SQL Server unless they are needed. SMB is needed if using and SMB File Share as a storage option, e.g. in a cluster or when using the SQL Server Filestream/FileTable feature with remote Win32 access enabled.

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 Configuration Manager tool and Policy-Based Management. SQL Server Configuration Manager provides granular configuration of server protocols. With SQL Server 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, and Named Pipes 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.
* Configure Extended Protection for both service binding and channel binding in SQL Server 2012 (refer to [SQL Server 2012 Books Online topic](http://msdn.microsoft.com/en-us/library/ff487261.aspx) for more information)

The dialog for configuring TCP/IP address properties such as port numbers and dynamic port assignment is shown in Figure 1.



**Figure 1   TCP/IP Addresses configuration page in SQL Server Configuration Manager**

In SQL Server 2008, 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, and TCP/IP 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**

* Enable Windows Firewall and limit the network protocols supported.
* Do not enable network protocols unless they are needed.
* Disable NETBIOS and SMB protocol unless specifically 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.
* Use extended protection in SQL Server 2012 if the client and operating system support it.
* 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.

## Auditing

SQL Server 2012 expands on the native auditing in the database engine added in SQL Server 2008. The SQL Server Audit feature maintains all the capabilities of the SQL Server 2005 auditing solutions and provides enhancements such as flexibility in audit data targets and granular auditing. In SQL Server 2012, Server Audit Specification is available in any edition of SQL Server, but SQL Server Database Audit Specification is available as an enterprise-only feature, although you can use Server Audit Specifications to audit all database-level events. Database Audit Specifications add the ability to do more granular auditing; to audit events from a particular user or schema for example. Earlier versions of SQL Server support login auditing, trigger-based auditing, and event auditing by using a built-in trace facility.

The SQL Server 2008 Audit feature is meant to replace trace-based auditing as the preferred auditing solution. This feature was designed with the following goals in mind:

* Security – The audit feature, and its objects, must be truly secure.
* Performance - Performance impact must be minimized.
* Management – The audit feature must be easy to manage.
* Discoverability - Audit-centric questions must be easy to answer.

SQL Server 2012 Audit can use a file as an auditing target but can also audit to the Windows Application Log or Windows Security Log. The Windows Security log is considered to be resistant to tampering and nonrepudiation, although its usage is generally controlled by a group policy object. Auditing to the Windows Security also enables integration with the Audit Collection Service (ACS) of the Microsoft System Center Operations Manager, which can securely collect audit information from the security logs of multiple machines and generate consolidated reports. SQL Server 2008 Audit metadata is defined using DDL and therefore can be managed using standard SQL Server permissions. Changes to the audit metadata, as well as enabling and disabling audit sessions, are also audited.

If SQL Server Audit is unable to write its audit events to the audit target, you can configure the audit object to shut down the server instance. This is necessary for meeting requirements of the Common Criteria to ensure that the server cannot operate without its activity being audited. In SQL Server 2012, you can also configure the audit object to fail the database operation associated with the audit object, but not shutdown the server. This allows maximum availability because un-audited operations continue to work. Additional implementation enhancements make auditing in SQL Server 2012 more resilient to transient network failures. If the instance cannot start because of a SQL Server Audit, it can be brought up using the –m or –f trace flags in order to issue the audit DDL necessary to fix the problem with auditing.

SQL Server Audit uses the Extended Events feature to minimize impact on performance. Extended Events are events that are built into the SQL Server code to have minimal impact. Audits can be written synchronously or asynchronously, with a configurable queue delay to accommodate a trade-off between database performance and possible audit record loss. Another way to minimize performance impact is to use more granular auditing. This feature, using Database Audit Specifications, is an Enterprise-only feature. SQL Server Audit allows defining audits at a database object and database login/user level, as well as providing a number of audit groups for convenience. At this time, auditing cannot be done at the column level. In SQL Server 2012, you can also audit user-defined audit events. These events are invoked by the application using the sp\_audit\_write system stored procedure.

Auditing can be configured through DDL as well as SQL Server Management Studio and SMO. This makes it easier to manage, because SQL Server standard management tools and permissions can be used. Three new database objects are used to manage the audit feature: SERVER AUDITs, SERVER AUDIT SPECIFICATIONs, and DATABASE AUDIT SPECIFICATIONs. An AUDIT object resides in the master database and defines where the audit information will be stored, a file rollover policy (if using file targets), a queue delay, and whether or not to shut down the instance if audit records cannot be written. A SERVER AUDIT SPECIFICATION or DATABASE AUDIT SPECIFICATION specifies what accesses and what principals to audit at a server or database level. One or more specifications are associated with an SERVER AUDIT to connect what is auditing to where the information is to be written.

Auditing information is stored in binary when written to file targets and can be read with a table-valued function fn\_get\_audit\_file(). Because access is through this table-valued function, it is easy to select and report on audit information using ordinary Transact-SQL code. Audit information written to the Windows log can be read using any of the Windows log-reading utilities, such as Windows Event Viewer. Both file and Windows log-based auditing information can also be read directly with SQL Server Management Studio.

For more information on the SQL Server auditing feature, see the whitepaper [Auditing in SQL Server 2008](http://msdn.microsoft.com/en-us/library/dd392015(v=SQL.100).aspx).

Password policy compliance is automatically enforceable through policy in SQL Server 2005 and above 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 breech. 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 and above also supports login triggers.

SQL Server 2005 introduced 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 2008 is larger than previous versions, more auditing events are available in SQL Server 2008 than in previous versions. To audit security events, use event-based auditing, specifically the events in the [security audit event category](http://msdn2.microsoft.com/en-us/library/ms191148.aspx) (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**. A Microsoft-supplied auditing script also needs to be run to set up profiler-based auditing. This script is available at the Microsoft download center at <http://download.microsoft.com/download/9/5/F/95FDD106-4E98-47B4-B676-7FDB9A403AF0/SQL_Server2008_EAL1_trace.sql>.

When C2 or Common Criteria 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. This script may be changed to use the SQL Server audit feature when SQL Server 2012 is Common Criteria certified.

**Best practices for auditing**

* Auditing is scenario-specific. Balance the need for auditing with the overhead of generating addition data.
* Use the SQL Server 2012 Audit feature for the most secure, performant, and granular auditing.
* 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 can be audited by using trace events or SQL Server Audit.
* Use WMI to be alerted of emergency events.

## Conclusion

Security is a crucial part of any mission-critical application. To implement security for SQL Server 2012 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 2012 security defaults and create a secure database configuration according to the Trustworthy Computing Initiative guidelines.

Additional References:

[SQL Server 2012 Security Public Web Site](http://www.microsoft.com/sqlserver/en/us/future-editions/mission-critical/denali-security-and-compliance.aspx)

[PCI Compliance with SQL Server](http://www.parentebeard.com/Uploads/Files/Deploying_SQL_Server_2008_Based_on_PCI_DSS.PDF)

[HIPAA Compliance with SQL Server](http://www.jeffersonwells.com/mssql2008hipaa)

[SQL Server in FIPS 140-2-compliance mode](http://support.microsoft.com/kb/955720)

[Security forum for SQL Server](http://social.msdn.microsoft.com/forums/en-US/sqlsecurity/threads/)

[SQL Server delivers industry-leading security](http://download.microsoft.com/download/9/8/5/98591434-2E0B-472F-8084-4C73661ABA4E/ITIC%20Microsoft%20SQL%20Server%20Security%20Final%209%208%2010.pdf)

[SQL Server Label Security Toolkit for Classified Databases Whitepaper](http://download.microsoft.com/download/8/8/0/880F282A-AA4E-4351-83C0-DFFA3B56A19E/SQL_Server_2012_RLS_and_CLS_White_Paper_January2012.docx)

[SQL Server Label Security Toolkit for Classified Databases Codeplex download](http://sqlserverlst.codeplex.com/)

[SQL Server Separation of Duties Framework](http://sqlserversod.codeplex.com/)

[SQL Server Certifications, Compliance, and Vulnerabilities](http://download.microsoft.com/download/3/7/2/37204386-5BAA-463E-8888-9140863531D6/SQLServer2008_ComplianceandCertifications.pdf)

[Auditing in SQL Server](http://msdn.microsoft.com/en-us/library/dd392015(SQL.100).aspx)

[Cryptography in SQL Server](http://msdn.microsoft.com/en-us/library/cc837966(SQL.100).aspx)

[Microsoft Security Compliance Manager](http://www.microsoft.com/downloads/en/details.aspx?FamilyID=5534bee1-3cad-4bf0-b92b-a8e545573a3e&displaylang=en)

[SQL Server 2008 R2 Security Best Practice – Operational and Administrative Tasks](http://download.microsoft.com/download/1/2/A/12ABE102-4427-4335-B989-5DA579A4D29D/SQL_Server_2008_R2_Security_Best_Practice_Whitepaper.docx)

[SQL Server 2008 R2 Separation of Duties for DBA](http://download.microsoft.com/download/D/2/D/D2D931E9-B6B5-4E3B-B0AF-22C749F9BB7E/SQL_Server_Separation_of_Duties_White_Paper_Jul2011.docx)

[SQL Server 2005 Security Best Practice – Operational and Administrative Tasks](http://www.microsoft.com/technet/prodtechnol/sql/2005/sql2005secbestpract.mspx)

For more information:

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

Did this paper help you? Please give us your feedback. Tell us on a scale of 1 (poor) to 5 (excellent), how would you rate this paper and why have you given it this rating? For example:

Are you rating it high due to having good examples, excellent screen shots, clear writing, or another reason?

Are you rating it low due to poor examples, fuzzy screen shots, or unclear writing?

This feedback will help us improve the quality of white papers we release.

[Send feedback](mailto:sqlfback@microsoft.com?subject=White%20paper%20Feedback:%20Do%20Not%20Pay%20Too%20Much%20For%20Your%20Database%20Licenses).