This white paper contains administrative and operational best practices that should be performed from a security perspective when using Microsoft SQL Server. The best practices cover operative instructions and example code snippets needed for DBA’s and Server Administrators.

Summary of Best Practices

- SQL Server should be hardened after the installation
- After the installation, use the SQL Server Configuration Manager tool in order to disable unnecessary features and services
- Install only required components
- Recent service packs and critical fixes should be installed for SQL Server and Windows
- Windows Authentication mode is more secure than SQL Authentication
- If there is still a need to use SQL Authentication – enforce strong password policy
- Disable the SA account and rename it. Do not use this account for SQL server management
- Change the default SQL Server ports associated with the SQL Server installation to keep hackers from port scanning the server.
- Hide SQL Server instances or disable the SQL Server Browser service.
- Remove BUILTIN\Administrators group from the SQL Server Logins.
- Enable failed & successful logging for SQL Server login attempts.

Reduce Attack Surface

SQL Server comes with several features that administrators can choose to install during the installation process:

- Database Engine
- Reporting Services
- Integration Services
- Analysis Services Engine
- Notification Services
- Documentation and Samples (Sample databases & codes)

It is good practice to install only the needed features and by doing so it will reduce the attack surface.
SQL Server Surface Area Configuration Tool

SQL Server 2005 contains configuration tools such as a system stored procedure called sp_configure or SQL Server Surface Area Configuration tool (for services and features) in order to enable/disable optional features as needed. Those features are usually installed as disabled by default. Here is the list of the features that can be enabled using the tool:

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

The command line version of the tool - sac.exe can be found at %Program Files%\Microsoft SQL Server\90\Shared. Using these tools makes it possible to import and export settings between several servers on the network. You must have sysadmin privilege in order to use this tool.

Following is an example of exporting all settings from the default instance of SQL Server on server A and importing them to server B:

```
sac out serverA.out -S serverA -U admin -I MSSQLSERVER
sac in serverA.out -S serverB
```

The following code snippet describes how to remove SQL Server components by using shell commands. At the command prompt, run the following command:

```
%ProgramFiles%\Microsoft SQL Server\90\Setup\Bootstrap\ARPWrapper.exe /Remove
```

Uninstall the SQL Server components one at a time until desired SQL Server components are uninstalled

**Stored Procedure – sp_configure**

Sp_configure is a system stored procedure which can be used instead of the Surface Area Configuration Tool in order to enable/disable the features in SQL Server. Following is an example of using the sp_configure in order to disable the xp_cmdshell extended stored procedure which enables a user to run shell commands on the server. The xp_cmdshell is turned off by default in SQL Server 2005 and on.

```
EXEC sp_configure 'show advanced options', 1
GO
RECONFIGURE
GO
EXEC sp_configure 'xp_cmdshell', 0
GO
RECONFIGURE
GO
```
**Additional Instructions**

- Do not install sample databases and sample code on production SQL Servers.
- If needed, install the samples only in the development and test environments.
- When upgrading from SQL Server 2000 to 2005 and on, it is important to review the configuration settings and turn off features such as `xp_cmdshell`, because the upgrade process does not change these settings by default.
- Turn off unnecessary services by setting them to disabled or manual startup.

**Windows and SQL Authentication Modes**

SQL Server supports two modes for validating connections and authenticating access to database resources: "Windows Authentication mode" and "SQL Server and Windows Authentication mode." Both of these authentication methods provide access to SQL Server and its resources.

**Windows Authentication Mode**

Windows Authentication mode is the default and recommended authentication mode. It leverages local accounts, Active Directory user accounts, and groups when granting access to SQL Server. In this mode, you, as the database administrator, are given the opportunity to grant domain or local server users access to the database server without creating and managing a separate SQL Server account. Another advantage of using Windows Authentication is management of password policy by the Active Directory or the local security policy.

**Mixed Mode - SQL Server & Windows Authentication**

The SQL Server authentication mechanism is based on accounts that are managed inside the database engine, including the password policy.

Mixed authentication (SQL Server and Windows Authentication mode) is still required if:
- There is a need to support legacy applications
- Applications require SQL Server authentication or clients from platforms other than Windows
- A need for separation of duties exists.

**Configuring SQL Server Authentication Modes**

To select or change the server authentication mode, follow these steps:

1. In SQL Server Management, right-click on a desired SQL Server and then click Properties.
2. Click on the Security tab.
3. On the Security tab, select the desired server authentication mode under Server Authentication and then click OK.
4. In the SQL Server Management Studio dialog box, click OK to acknowledge the need to restart SQL Server.
5. In Object Explorer, right-click on a desired server and then click Restart. If the SQL Server Agent is running, it also requires a restart.

Using Windows Authentication is a more secure choice. However, if mixed mode authentication is required, you must leverage complex passwords and the SQL Server password and lockout policies to further secure the instance.
Here is an example of password policy for SQL Server logins:

- The password must contain uppercase & lowercase letters.
- The password must contain numbers & alphanumeric characters.
- The password must contain non-alphanumeric characters such as &, ^, %, $, etc.
- Do not use common known passwords that are easy to guess such as: admin, password, sa, administrator, sysadmin, etc.
- Passwords should be at least 8 characters long.
- SQL Server 2005 and on does not allow blank passwords for the SA account.
- If you are using an earlier version of SQL Server, set a strong password for SQL Server logins and also for the SA login according to the password policy.

Note: If Windows Authentication mode is selected during installation, the SA login is disabled by default. If the authentication mode is switched to SQL Server mixed mode after the installation, the SA account is still disabled and must be manually enabled. It is a best practice to reset the password when the mode is switched.

**Additional Instructions**

- Use Windows Authentication mode when it is possible.
- Use Mixed Mode Authentication only for legacy applications and non-Windows users.
- When using Mixed Mode Authentication – Potential attackers are aware of the SA login and this makes hacking one step easier if they want to take control of this powerful account. Thus, in this mode the SA account must be renamed to a different account as follows:

```
USE MASTER
ALTER LOGIN sa DISABLE;
GO
ALTER LOGIN sa WITH NAME = [WinnerUser];
```

Note: Before renaming the SA account, verify that another login with administrative privileges exists and you are able to login to the instance in order to have access to the SQL Server.

- Establish a strong password policy for the SA login and change the password periodically.
- Do not manage SQL Server by using the sa login; assign sysadmin privileges to another user or group.

**Administrative Privileges**

Privileges with elevated permissions in SQL Server include:

- SQL Server SA build-in login (relevant if it is enabled)
- Members of the sysadmin (BUILT-IN\Administrators) SQL Server role.
- Logins with CONTROL SERVER permission

**Instructions**

- Administrator privileges should be used only when they are really needed
- Define only a minimal number of administrators.
- Allocate different administrative accounts if there is more than one administrator
- Provision admin principals explicitly
- Avoid dependency on the “BUILTIN\Administrators” Windows group
Here is Transact-SQL (T-SQL) syntax for removing the BUILTIN\Administrators Windows Group from a SQL Server instance in case of this group exists from a previous versions of SQL Server. This code should be executed on each SQL Server instance installed in the organization as long as it is not needed by another application such as Clustering:

```sql
USE MASTER
IF EXISTS (SELECT * FROM sys.server_principals
WHERE name = N’BUILTIN\Administrators’)
DROP LOGIN [BUILTIN\Administrators]
GO
```

**Defining Database Ownership**

A SQL Server instance can contain many databases which are created by logins who are also database owners -DBO (by default) as it can be seen in the following image: The login workshop created the workshop database and is a member of db_owner database role.

Also members of server role sysadmin are also database owners, who have the db_owner role in every user database. Members of the db_owner role have approximately the same privileges as the dbo use. For example the SA user is a member of sysadmin server role as it can be seen in the following image:

The SA user is also a member of the db_owner role.

Thus several best practices should be implemented regarding these special ownerships:

- Minimize the number of users that have the db_owner role for each database.
- Have distinct owners for databases; not all databases should be owned by SA or by any other user in the sysadmin server role.

**Disabling certain system stored procedures**

SQL Server ships with various system stored procedures such as xp_cmdshell or sp_send_dbmail that interact with the operating system or execute code outside of a normal SQL Server permission and may constitute additional security risks. Thus such stored procedures should be treated special.

In SQL Server it is possible to enable/disable these features in the Surface Area Configuration tool – SAC - (as was mentioned before) which can be found at Start -> All Programs -> Microsoft SQL Server 2005 -> Configuration Tools -> Surface Area Configuration tool. The following features should stay disabled (which are disabled by default):

- Disable xp_cmdshell unless it is absolutely needed.
- Disable COM components once all COM components have been converted to the SQL CLR.
- Disable both mail procedures (Database Mail and SQL Mail) unless you need to send mail from SQL Server.
**Additional Instructions**

- Use SQL Server Surface Area Configuration to enforce a standard policy for extended procedure usage.
- Document each exception to the standard policy.
- Do not remove the system stored procedures by dropping them.
- Do not DENY all users/administrators access to the extended procedures.

**Notes:** Some system stored procedures, such as procedures that use SQLDMO and SQLSMO libraries, cannot be configured by using SQL Server Surface Area Configuration. They must be configured by using sp_configure or SQL Server Management Studio (SSMS) directly.

In SQL Server 2008, the SAC was replaced by Policy Based Management framework. This could be accessed from the SSMS under the Management option. It allows you to define and enforce policies for configuring and managing SQL Server across the enterprise.

---

**Hardening the network connectivity**

**Hardening SQL Server Ports with SQL Configuration Manager**

A default installation of SQL Server uses TCP port 1433 for client requests and communications. These ports are well known and are a common target for hackers. Therefore it is recommended to change default ports associated with the SQL Server installation.

Follow these steps to change the default port using SQL Server Manager Configuration tools:

1. Choose Start, All Programs, Microsoft SQL Server, Configuration Tools, SQL Server Configuration Manager.
2. Expand the SQL Server Network Configuration node and select Protocols for the SQL Server instance to be configured.
3. In the right pane, right-click the protocol name TCP/IP and choose Properties.
4. In the TCP/IP Properties dialog box, select the IP Addresses tab.
5. There is a corresponding entry for every IP address assigned to the server. Clear the values for both the TCP Dynamic Ports and TCP Port for each IP address except for the IP addresses under IPAll.
6. In the IPAll section for each instance, enter a new port that you want SQL Server to listen on.
7. Click Apply and restart the SQL Server Services.

**Hiding an SQL Server Instance from Broadcasting Information**

The SQL Server Browser service broadcasts SQL Server information on the network. Because of this, attackers can use SQL Server clients to browse the current infrastructure and retrieve a list of running SQL Server instances. Thus it is strongly recommended to hide SQL instances from being shown in the network as follows:

1. Choose Start, All Programs, Microsoft SQL Server, Configuration Tools, SQL Server Configuration Manager.
2. Expand the SQL Server Network Configuration node and select Protocols for the SQL Server instance to be configured.
3. Right-click Protocols for [Server\Instance Name] and then choose Properties.
4. On the General tab for the Hide Instance option, select the ‘Yes’ option.
5. Click OK and restart the services for the change to take effect.

Additional Instructions

- Allow only network protocols that are needed.
- CONNECT permission should be granted only on endpoints to logins that need to use them.
- If there is a need to work with SQL Server Login, install an SSL certificate from a trusted certificate authority (CA) rather than using SQL Server’s self-signed certificates.
- Avoid the exposure of SQL Server directly to the public internet.

Encryption principles

Sensitive information such as Credit card numbers, SSN, Salaries, etc. stored in a SQL Server database should be encrypted.

SQL Server includes encryption capabilities as described below:

Cell Level Encryption – Supported in SQL Server 2005 (Supported also in 2008)

SQL Server has an encryption hierarchy, as shown below, that needs to be followed in order to properly support the native encryption capabilities.

The top-level resource in the SQL Server encryption hierarchy is the Service Master Key, which is encrypted by the Windows Data Protection API and created the first time a newly-created key needs to be encrypted. Second in line is the Database Master Key. This key can be used to create certificates and asymmetric keys. Next come certificates and asymmetric keys. Both can be used to create symmetric keys or encrypt data directly. Last in line are symmetric keys, which can also be used to encrypt data.

Here are the steps for encrypting information on a cell level:

- **Create a Database Master Key** - Before creating certificates (assuming the Database Master Key is used to create the certificate) or other keys can be generated, a Database Master Key must be created:
  
  ```sql
  USE <DatabaseName>
  CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<Password>'
  ```

- **Backup Database Master Keys** - Once a Database Master Key is created, you should back it up to a secure location, preferably offsite. To back up a Database Master Key, use the following syntax:
  
  ```sql
  USE <DatabaseName>
  BACKUP MASTER KEY TO FILE = '<FolderLocation>'
  ENCRYPTION BY PASSWORD = '<Password>'
  ```
• **Create a certificate** - For creating symmetric keys for data encryption or to encrypt the data directly

```
USE <DatabaseName>
CREATE CERTIFICATE certXXXXX WITH SUBJECT='XXXXXX Certificate',
EXPIRY_DATE = '03/09/2018'
```

• **Backup the Certificate**

```
USE <DatabaseName>
BACKUP CERTIFICATE certXXXXX TO FILE = '<FileLocation>',
ENCRYPTION BY PASSWORD='<Password>'
```

• **Encrypt the data via certificate**

```
INSERT INTO [DatabaseName].[SomeTable]
values( N'data encrypted by certificate 'ThisIsSomething'',
EncryptByCert(Cert_ID('certXXXXX'), @cleartext));
GO
```

• **TDE – Transparent Data Encryption in SQL Server 2008 (Database Level Encryption)**

TDE provides real time encryption of data and log files. It is important to mention that this is **database level encryption**. Data is encrypted before it is written to disk and also data is decrypted when it is read from disk. The “transparent” aspect of TDE is that the encryption is performed by the database engine and SQL Server clients are completely unaware of it. There is absolutely no code that needs to be written to perform the encryption and decryption. There are a couple of steps to be performed to prepare the database for TDE, and then the encryption is turned on at the database level via an ALTER DATABASE command.

With TDE the backup files are also encrypted when using just the standard BACKUP command once encryption is turned on for the database.

Here are the steps that enable TDE in SQL Server:
- **Create a Master Key** - A master key is a symmetric key that is used to create certificates and asymmetric keys. Execute the following script to create a master key:

```sql
USE master;
CREATE MASTER KEY
ENCRYPTION BY PASSWORD = 'Create^A#LongPassw@rd123';
GO
```

Note that the password should be a strong with alpha, numeric, upper, lower, and special characters and you have to backup (i.e. use BACKUP MASTER KEY) the key and store it in a secure location.

- **Create a Certificate** – Certificates can be used to create symmetric keys for data encryption or to encrypt the data directly. Execute the following script to create a certificate:

```sql
USE master;
CREATE CERTIFICATE TDECert
WITH SUBJECT = 'TDE Certificate'
GO
```

Note that certificates also need to be backed up (i.e. use BACKUP CERTIFICATE) and stored in a secure location.

- **Create a Database Encryption Key** - A database encryption key is required for TDE. Execute the following script to create a new database and a database encryption key for it*:

```sql
CREATE DATABASE mssqltips_tde
GO
USE mssqltips_tde;
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDECert
GO
```

In order to work with TDE the encryption key must be encrypted by a certificate (a password will not work) and the certificate must be located in the master database.

- **Enable TDE** - The final step required to implement TDE is to execute the following script*:

```sql
ALTER DATABASE mssqltips_tde
SET ENCRYPTION ON
GO
SELECT [name], is_encrypted FROM sys.databases
GO
```

Additional Instructions

- Encrypt high-value and sensitive data
- Use symmetric keys to encrypt data, and asymmetric keys or certificates to protect the symmetric keys
- Password-protect keys and remove master key encryption for the most secure configuration
- Always back up the service master key, database master keys, and certificates by using the key-specific DDL statements
- Always back up your database to back up your symmetric and asymmetric keys

SSL Configuration in SQL Server

Microsoft SQL Server can use Secure Sockets Layer (SSL) to encrypt data that is transmitted across a network between an instance of SQL Server and a client application. Enabling SSL encryption increases the security of data transmitted across networks between instances of SQL Server and applications.

Instructions – Configuring SSL for SQL Server

1. Install a certificate in the Windows certificate store of the server computer.
2. Click Start, in the Microsoft SQL Server program group, point to Configuration Tools, and then click SQL Server Configuration Manager.
3. Expand SQL Server Network Configuration, right-click the protocols for the server you want, and then click Properties.
4. On the Certificate tab, configure the Database Engine to use the certificate.
5. On the Flags tab, view or specify the protocol encryption option. The login packet will always be encrypted.
6. When the ForceEncryption option for the Database Engine is set to Yes, all client/server communication is encrypted and clients that cannot support encryption are denied access.
7. When the ForceEncryption option for the Database Engine is set to No, encryption can be requested by the client application but is not required.
8. SQL Server must be restarted after you change the ForceEncryption setting.

Auditing Mechanism in SQL Server

SQL Server security auditing monitors and tracks activity to log files that can be viewed through Windows application logs or SQL Server Management Studio. SQL Server offers the following four security levels with regards to auditing:

- None—Disables auditing (no events are logged)
- Failed Logins Only—Audits all failed login attempts
- Successful Logins Only—Audits all successful login attempts
- Both Failed and Successful Logins—Audits all login attempts

The default mode is: Failed Logins Only. Thus, it is recommended to set the auditing mode to be Both Failed and Successful Logins.
Configuring SQL Server Security Logs for Auditing

To configure security login auditing for both failed and successful logins, follow these steps:

1. In SQL Server Management Studio, right-click on a desired SQL Server and then click Properties
2. On the Security tab under Login Auditing, select the desired auditing criteria radio button, such as Both Failed and Successful Logins, and then click OK
3. Restart the SQL Server Database Engine and SQL Server Agent to make the auditing changes effective

Additional Instructions

• Auditing is scenario-specific. Balance the need for auditing with the overhead of generating additional data.

• Audit successful logins in addition to unsuccessful logins if you store highly sensitive data.

• Enable C2 auditing or Common Criteria compliance only if required by selecting the appropriate checkbox. Those options should be selected only if there is a need to comply with these security standards.

In order to configure the C2 audit, you have to activate (in Query Analyzer or osql.exe) and run the following (Command example):

```sql
EXEC sp_configure 'show advanced option', '1'
go
RECONFIGURE
go
EXEC sp_configure 'c2 audit mode','1'
go
RECONFIGURE
```

Tools for analyzing best practices of SQL server hardening

The SQL Server Best Practices Analyzer (BPA) gathers data from Microsoft Windows and SQL Server configuration settings. The Best Practices Analyzer uses a predefined list of SQL Server recommendations and best practices to determine if there are potential security issues in the database environment.

For SQL Server 2012

**Microsoft SQL Server Security Best Practices**

By David Maman, GreenSQL CTO

---

### Additional Instructions

- Run periodically the BPA in order to find security issues.

### Patching and updates

Security updates and patches are constantly being released by Microsoft. It is advantageous to install these updates for SQL Server and the operating system. These patches can be manually downloaded and installed, or they can be automatically applied by using Microsoft Update.

### Additional Instructions

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