Anyone who's been around the IT industry knows that security is a hot topic. After all, a company's data is one of the most precious assets it has and protecting that data is critical. As I was planning what to write for this security issue, I tried to determine which security feature should have an entire article dedicated to it. But then I started to think about the burgeoning numbers of "involuntary DBAs" (those IT pros who inadvertently wind up responsible for a SQL Server instance) and the great response we had to the [Top Tips for Effective Database Maintenance](https://technet.microsoft.com/en-us/magazine/2008.08.database.aspx) article from the August 2008 issue. It struck me that what I should do is write an article that covers common SQL Server security issues—a sort of primer to better security for your SQL Server instances.

An article that describes all the security problems you should be looking at would be a whole magazine in itself, so I canvassed my fellow SQL Server MVPs (and my wife) for input as to what I should include. I settled on providing the top 10 security areas I think you should worry about if you're not a security-savvy DBA and suddenly find yourself responsible for a SQL Server instance and associated application. (Remember that this is by no means an exhaustive list.) For each issue, I've given a short overview of the problem, advice on how to mitigate it, and a link to more detailed information.

All the links to Books Online are for the SQL Server 2008 versions, but each Web page links to the SQL Server 2005 version as well. Wrapping up the article, I provide links to the main white papers and Books Online sections for security for SQL Server 2005 and SQL Server 2008.

Physical Security

It's imperative that you don't just consider SQL Server security itself, but all the layers that must be traversed to get to SQL Server in the first place. For this concept of layered security, often called "defense in depth," you don't just secure one layer, but consider the entire system.

The most obvious first consideration is the physical security of the machine running SQL Server—this layer, however, is often overlooked or is the subject of complacency. I'm not referring simply to whether the machine can be stolen or not, but also to physical access to things like the storage system hosting the database files, the backup tapes, and any servers hosting redundant copies of the database. Only those people with an actual need to physically touch these objects should have access to them; anyone who needs temporary access should be accompanied and monitored.

A less obvious consideration is the security of the desktops of the people who have high-privileged access to SQL Server. If someone has sysadmin access to SQL Server but they leave their Windows desktop unlocked, then all the security in the world isn't going to prevent someone walking past the unattended system from potentially accessing sensitive data. A more insidious problem would be if someone walked past and changed some data—for instance, a dishonest employee who knows the schema of the human resources database and tries to undetectably change a salary.

There's a very interesting TechNet white paper (including a slide-deck and webcasts), titled "[Physical Security at Microsoft](https://technet.microsoft.com/en-us/library/cc537553.aspx)," which describes how Microsoft manages the physical security of its many servers.

Network Security

Although your servers may be physically inaccessible, they're most likely connected to a network of some kind. This could be just an isolated company LAN with no outside connections, or it could be a direct connection to the Internet. No matter what the situation, there are some things you need to consider:

* Ensure that the Windows server has proper network security configured.
* Decide which network protocols to allow, and disable any that are not required.
* Ensure there is a firewall set up (such as Windows Firewall) and configure it to allow access to SQL Server (as shown in **Figure 1**).
* Decide whether to encrypt connections to SQL Server and configure appropriately.
* If Kerberos will be used, register a Server Principal Name. Kerberos is an authentication mechanism that underpins Windows authentication (which I describe later in this article), but it is poorly understood. A clear and concise explanation was provided by Rob Greene, a Support Escalation Engineer, in the blog post "[Kerberos for the Busy Admin](http://blogs.technet.com/askds/archive/2008/03/06/kerberos-for-the-busy-admin.aspx)." I recommend checking it out.
* Decide whether to use the SQL Server Browser Service to help clients find installed SQL Server instances, and decide whether you want to hide some instances. Hiding an instance means client applications and users will need to know the connection details of the SQL Server instance, but it prevents people from trawling the network to look for SQL Server instances.

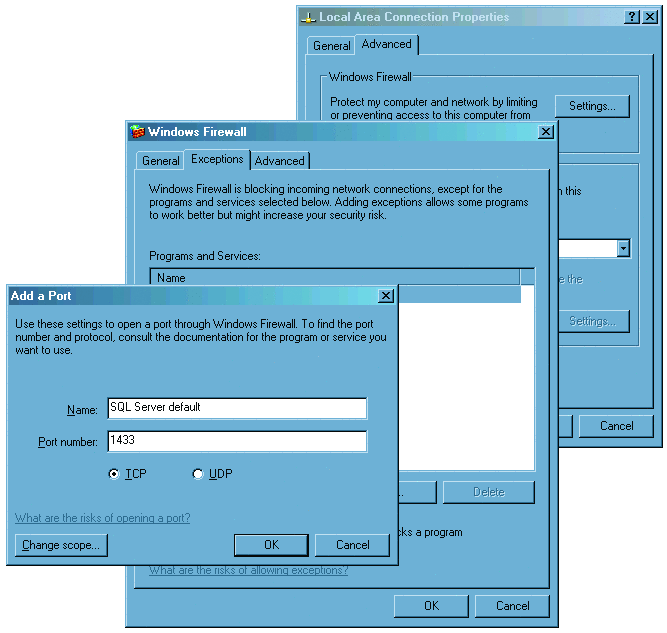


Figure 1 **Configure Windows Firewall to allow TCP access to SQL Server**

All these tasks (and more) are explained in SQL Server 2008 Books Online, starting with the topic [Server Network Configuration](https://technet.microsoft.com/library/ms189083.aspx). There's also a good section on [Client Network Configuration](https://technet.microsoft.com/library/ms190611.aspx).

Attack Surface Minimization

The more services and features that are enabled, the more attack surface or surface area there is for bad guys to attack. SQL Server 2005 took the "off by default" strategy—features with security implications are disabled by default and are only enabled by the DBA when required. (This process of enabling and disabling services is commonly called surface area configuration.)

A prime example of a feature that you may want disabled is xp\_cmdshell, which provides a way of executing commands on the host Windows system from within the context of a SQL Server instance. If an intruder compromises the SQL Server instance, and the SQL Server service account has elevated Windows privileges, xp\_cmdshell can be used to gain access to the Windows system, too.

There are a variety of methods for surface area configuration. Both SQL Server 2005 and SQL Server 2008 support the SQL Server Configuration Manager for configuring services and network protocols. Both also support the sp\_configure stored procedure for configuring database engine features and options. As an example, this code will disable the xp\_cmdshell feature:

-- To allow advanced options to be changed EXEC sp\_configure 'show advanced options', 1; GO -- To update the currently configured value for -- advanced options RECONFIGURE; GO -- To disable xp\_cmdshell EXEC sp\_configure 'xp\_cmdshell', 0; GO -- To update the currently configured value for this -- feature RECONFIGURE; GO

The graphical methods of configuring database engine options and features are quite different between the two versions. SQL Server 2005 introduced the SQL Server Surface Area Configuration (SAC) tool. This allows you to easily make changes. **Figure 2**, for instance, shows how you can disable xp\_cmdshell using SAC.

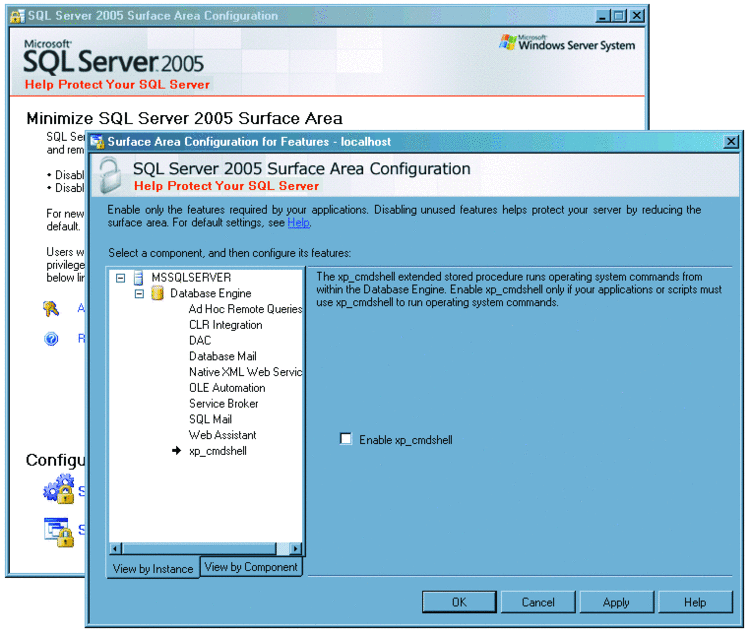


Figure 2 **Disabling xp\_cmdshell using SAC in SQL Server 2005**

In SQL Server 2008, SAC was completely removed and the functionality subsumed by the Policy-Based Management feature. You can find more information on this in the SQL Server 2008 Books Online topic "[Administering Servers by Using Policy-Based Management](https://technet.microsoft.com/library/bb510667.aspx)." **Figure 3** illustrates creating a condition to check that xp\_cmdshell is disabled. Once configured, policies can target both SQL Server 2005 and SQL Server 2008 instances and can even be "applied," allowing you to essentially change the setting with one click.

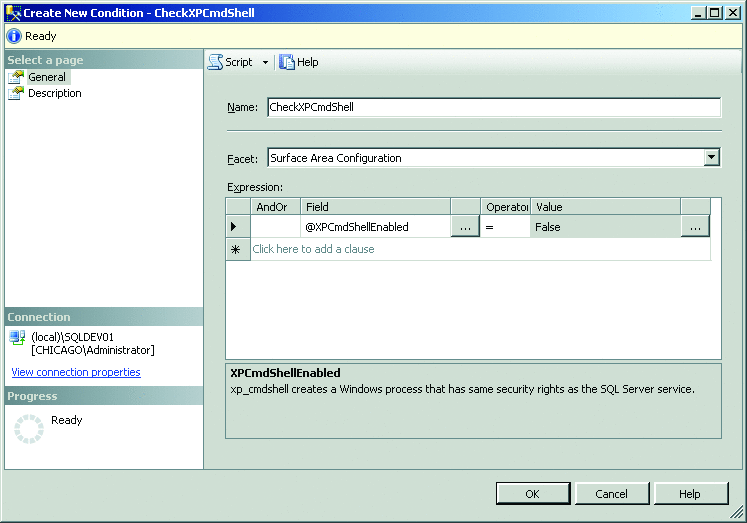


Figure 3 **Creating a Policy-Based Management condition in SQL Server 2008**

Service Accounts

SQL Server runs as one or more services on Windows, and each service needs to have a Windows account that is used to run the service. The account needs to have access to various resources on the Windows system (such as the network and various file system directories). The best practice is for the account to have the least possible privileges required to allow SQL Server to operate correctly. This is part of what is called the principle of least privilege, which states that a system can be made more secure by granting a user or process only those privileges required and nothing more.

As such, a SQL Server service account should not be a high-privileged account (such as Local System or Local Administrator) because if SQL Server is compromised, there is the potential for the Windows system to also be compromised. The services typically run under a built-in account called Network Service, but if SQL Server requires access to other domain resources, a new domain user account should be created with the minimum privileges and resource accesses required. The SQL Server 2008 Books Online topic "[Setting Up Windows Service Accounts](https://technet.microsoft.com/library/ms143504.aspx)" provides a comprehensive list of service accounts, required privileges, and resources. Note that if you must change a service account (or anything about it), you should always use the SQL Server Configuration Manager to ensure that all the necessary configuration changes are properly made.

Restricting Use of Administrator Privileges

The more exposure the SA login or the sysadmin fixed server role has, the more potential there is for a security breach—or an accident. There are a few best practices to follow.

Minimize the number of people who have access to the SA account and restrict the membership of the sysadmin role (and other privileged roles) so only those people who really need sysadmin privileges have them. Don't give out the SA password to anyone who wants temporary access to SQL Server or to a SQL user who wants to perform some quick task. In these situations, create a new SQL login and grant whatever privilege is required.

It's better to have people as members of the sysadmin role rather than using the SA account. This way, you can remove a user's login without having to change the SA account password. If you take over an old server and you have no idea who had access to it before, change the SA password so you're in control.

One issue that provokes debate is whether to remove the Windows BUILTIN/Administrators group from the sysadmin role (it's added by default). Should a Windows admin be able to query the HR database? Or does your company explicitly trust all administrators to have integrity and honesty? If you decide to remove it, proceed with care. This is especially true in a clustered environment where if you don't take the correct steps, your SQL Server may not start. For more information on this and other clustering issues, see the Knowledge Base article "[Clustered SQL Server Dos, Don'ts, and Basic Warnings](http://support.microsoft.com/kb/254321)."

For those who do have sysadmin access, encourage them not to log in with high privileges unless absolutely necessary—give each of these users two logins, one privileged and one not. Using the nonprivileged account by default helps minimize the potential for costly mistakes, and also reduces the likelihood that an unlocked Windows terminal will have a window with sysadmin privileges open on it. Remember: defense in depth!

The last point I'll make here is to avoid having applications that require sysadmin privileges. Unfortunately, this is a common and unavoidable bad practice with some applications, but you should avoid this practice in homegrown applications and complain to any application developers that require sysadmin privileges.

Authentication

There are two authentication modes available: Windows Authentication and mixed-mode (which is Windows authentication combined with SQL Server Authentication).

Windows Authentication uses network/domain accounts to validate the Windows account being used to connect to SQL Server. If this is selected during setup, the SA account is created with a randomly generated password but effectively disabled. Immediately after installation, you should change the SA password to a strong and secure password but continue to leave the account disabled unless absolutely necessary.

SQL Server Authentication relies on each user having a defined SQL Server account and password stored in SQL Server. And, of course, this means that the SA account is enabled and must have a defined password. With SQL Server authentication, users have at least two user names and passwords (one for the network and one for SQL Server). It is generally recommended that you only use Windows Authentication when possible, as it is a more secure solution. The Books Online topic "[Choosing an Authentication Mode](https://technet.microsoft.com/en-us/library/ms144284.aspx)" explains this in more detail, along with how to change the authentication mode.

If SQL Authentication is used, then all users should have a strong password—one that is complex enough to be resistant to being guessed by a password-cracking program. This is especially important for high-privileged accounts, such as SA and members of the sysadmin role. (One of the most common yet worst practices used to be to have a blank SA password. Fortunately, as of SQL Server 2000 SP3, that is not possible.)

Passwords should also be changed regularly and your company should publish corporate policies to help employees use secure and strong password best practices. For an overview of best practices for creating and protecting strong password, see the article "[Strong Passwords: How to Create and Use Them](http://microsoft.com/protect/yourself/password/create.mspx)." You can incorporate these policies into a corporate policy.

If SQL Server 2005 or SQL Server 2008 is running on Windows Server 2003 or later, it can make use of the Windows password policy mechanisms to enforce complexity and expiration policies. The SQL Server 2008 Books Online section called "[Password Policy](https://technet.microsoft.com/library/ms161959.aspx)" has information about support for strong passwords and various password policies in SQL Server.

Authorization

As I've already mentioned, one of the tenets of securing systems is to use the principle of least privilege. While this applies directly to admin-level privileges, it also applies to permissions for regular database users. A user in one database should (probably) not be able to access data in another database. The owner of one set of tables should not be able to mess with someone else's tables. Users should only be able to access the data they are supposed to access, and even then they should only be able to perform required actions on the data (for instance, SELECT but not UPDATE or DELETE).

All of this can be accomplished within SQL Server by a comprehensive, hierarchical permissions system where users or roles (called principals) are granted or denied certain specific permissions on certain resources (called securables) such as an object, schema, or database. An overview of the SQL Server permissions hierarchy is illustrated in **Figure 4**. This also implies that you follow the principle of least privilege. For example, don't make all database developers members of the db\_owner role. Restrict permissions to the public role and only grant permissions to the lowest level (user or role) to minimize direct access. A full discussion of best practices for permissions is beyond the scope of this article, but SQL Server 2008 Books Online includes a section called "[Identity and Access Control (Database Engine)](https://technet.microsoft.com/en-us/library/bb510418.aspx))" that offers drill-downs into all the concepts.

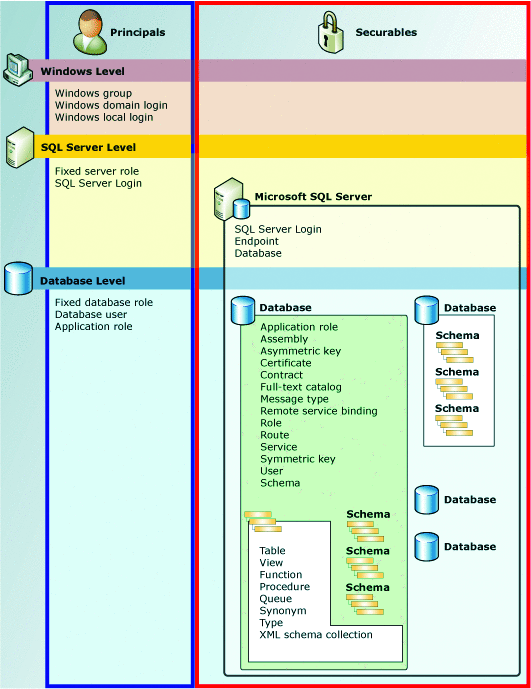


Figure 4 **The SQL Server permissions hierarchy**

To allow more granular permissions and better separation of roles within a database, SQL Server 2005 introduced user-schema separation, where schemas are independent of database users and are just containers of objects. This allows many behavioral changes, including more fine-grained precision for managing permissions. (Four-part naming follows the format server.database.schema.object in SQL Server 2005 and SQL Server 2008—the format was previously server.database.owner.object.)

For example, a schema can be created with database developers given CONTROL permissions. They can CREATE, ALTER, and DROP any objects within the schemas they control but they have no implied permissions to any other schemas within the database and they no longer need db\_owner rights for database development. In addition, user-schema separation allows database users to be dropped without having to recode all of the related objects or the application with a new user name—the objects are members of a schema and they are no longer tied to the object's creator.

It's a best practice to use schemas for object ownership because of these reasons. More information can be found in SQL Server 2008 Books Online in the topic "[User-Schema Separation](https://technet.microsoft.com/library/ms190387.aspx)."

Another way of preventing users from doing things they're not supposed to do is to disallow direct access to your base tables. This can be done by providing stored procedures and functions that are used to encapsulate, control, and isolate operations like updates and deletes, and by providing views that allow controlled and optimal selects.

SQL Injection

One of the most common methods of gaining unauthorized access to data is to use a SQL injection attack. SQL injection can take many forms, but the primary approach takes advantage of code that uses dynamically constructed strings and "injects" unexpected code into the construct. For example, the following injection attack takes advantage of poorly written logic for validating user input to trick SQL Server into accepting input by including escape characters in the input string. Although contrived, this example highlights what can happen when code dynamically constructs strings using input that is not thoroughly validated:

DECLARE @password VARCHAR (20); DECLARE @input VARCHAR (20); DECLARE @ExecStr VARCHAR (1000); SELECT @password = 'SecretSecret'; -- assume application gets input 'OR''=' SELECT @input = '''OR''''='''; SELECT @ExecStr = 'IF ''' + @password + ''' LIKE ''' + @input + ''' PRINT ''Password Accepted'''; EXEC (@ExecStr); GO

If you run this code, it will print the phrase "Password Accepted" even though the user input clearly did not match the password string. The user input contained an escape sequence that changed the Transact-SQL logic because the input was not properly parsed and checked.

SQL injection should not be a problem for a well-written application and there are some specific tricks (like using QUOTENAME to delimited identifiers). But if you inherit an old application (or maybe a homebrew project that became a corporate application), you should specifically test to see whether it is vulnerable to SQL injection attacks. There are a variety of techniques available to mitigate SQL injection attacks. SQL Server 2008 Books Online has a comprehensive section, starting with the aptly named topic "[SQL Injection](https://technet.microsoft.com/library/ms161953.aspx)."

Disaster Recovery

How much you need to worry about this depends on what your downtime and data loss requirements are. (If you haven't already defined these requirements, you should!) Security problems can occur at many levels. There are some concerns when disaster recovery involves failover to another SQL Server or the need to restore a database containing encrypted data.

In the first case, problems occur when the logins necessary to access the database have not been duplicated on the failover server—for instance, when using log shipping or database mirroring. If the database fails over to the mirror instance and the application tries to connect using a specific login that does not exist on the mirror server, the application will receive the 18456 error "login failed". The logins are part of the application ecosystem and must be defined on the instance hosting the database. Knowledge Base article 918992 "[How to Transfer the Logins and the Passwords Between Instances of SQL Server 2005](http://support.microsoft.com/kb/918992)" explains how to do this, and I will discuss troubleshooting error 18456 in a moment.

In the second case, problems occur when the database backup contains encrypted data and the encryption key (or keys) used to encrypt the data were not backed up or are not available in the SQL Server instance where the database is being restored. The best case here is that only some of the data in the database is encrypted and so only that subset of data cannot be accessed. The worst case scenario is that the entire database has been encrypted using Transparent Data Encryption (TDE) in SQL Server 2008. In that case, if the server certificate used to protect the database encryption key was not backed up or is not available, the entire database cannot be restored and the following errors will be returned:

Msg 33111, Level 16, State 3, Line 1 Cannot find server certificate with thumbprint '0xFBFF1103AF133C22231AE2DD1D0CC6777366AAF1'. Msg 3013, Level 16, State 1, Line 1 RESTORE DATABASE is terminating abnormally.

Of course, that's the point of TDE—that someone happening across a stray backup of the encrypted database can't restore it and access the sensitive data. But if the data is yours, and you need to access it, then you must have the server certificate available or the data is lost.

Encryption is a huge topic in itself and has comprehensive coverage in SQL Server 2008 Books Online, starting with the "[SQL Server Encryption](https://technet.microsoft.com/en-us/library/bb510663.aspx)" section. You can also watch me demonstrate TDE and successfully restore to a second instance in an accompanying video screencast available at .com/tips.

Auditing

One of the most important things you should do to improve the security of a system is to implement auditing. With this, you'll know who's doing what. This may even be mandatory, depending on the nature of your business.

At the very least, you should audit both failed and successful logins so that you can tell if, for instance, five failed login attempts were followed by a successful one. Then you know when someone is trying to break into your SQL Server instance (and with which login). **Figure 5** shows configuring login auditing through the Server Properties dialog box in SQL Server 2005 Management Studio. Failed logins are audited as message 18456 in the error log and the error state gives the reason for the failure. The best description I could find of the different states, as well as a long discussion about troubleshooting them, is in a post from Il-Sung Lee on the SQL Protocols blog titled "[Understanding 'Login Failed' (Error 18456) Error Messages in SQL Server 2005](http://blogs.msdn.com/sql_protocols/archive/2006/02/21/536201.aspx)."

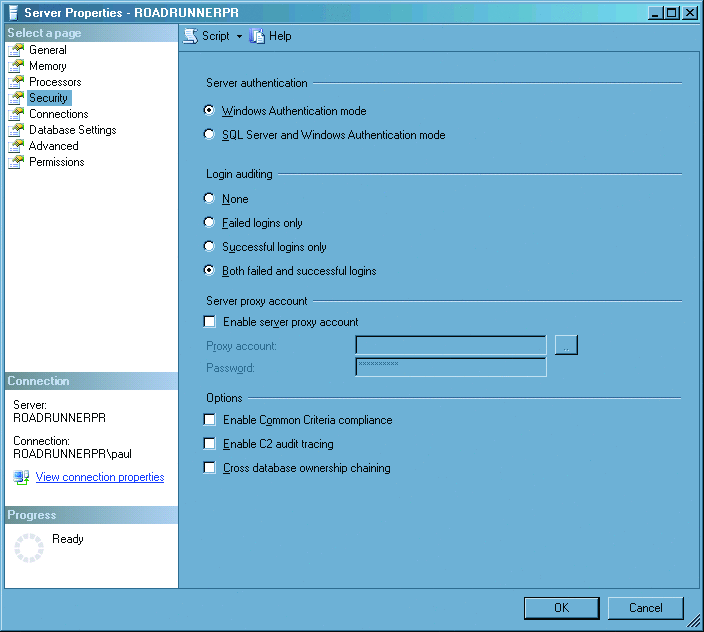


Figure 5 **Confi guring login auditing in SQL Server 2005 Management Studio**

Comprehensive auditing of all actions is difficult in SQL Server 2005 (and well beyond the scope of this article). It involves various triggers and SQL Trace. In SQL Server 2008, auditing was vastly simplified with the introduction of a new feature: SQL Server Audit. This is covered in a recent, very detailed white paper titled "[Auditing in SQL Server 2008](https://technet.microsoft.com/library/dd392015.aspx)." In the accompanying video screencast, I demonstrate SQL Server Audit. You can get details on the other auditing tools in the SQL Server 2008 Books Online "[Auditing (Database Engine)](https://technet.microsoft.com/library/cc280526.aspx)" section.

Summary

That's a lot of topics and a lot of links, but that was the point of this article. I wanted to provide an overview of important security topics that you must consider when you're a DBA who isn't used to dealing with security.

There are some tools that can help you check your system for common security vulnerabilities. The first is the Microsoft Baseline Security Analyzer (MBSA) utility that will check for Windows, network, file system, and even some SQL Server 2000 and SQL Server 2005 security issues. The latest version is [Microsoft Baseline Security Analyzer 2.1](https://technet.microsoft.com/security/cc184923.aspx). Again for SQL Server 2000 and SQL Server 2005 only, there is a SQL Server–specific tool called the [Best Practices Analyzer](http://go.microsoft.com/fwlink/?LinkId=145394) (BPA). This uses a pre-defined list of rules to check your SQL Server configuration and flags any problems (for instance, a blank SA password).

Neither of these tools works with SQL Server 2008. In fact, BPA will not be released for SQL Server 2008 at all and has been replaced, along with the short-lived SQL Server 2005 Surface Area Configuration tool, by the new Policy-Based Management feature I discussed earlier. Part of the reason for this replacement is that BPA and SAC were read-only tools to help you find problems—they could not fix problems. Policy-Based Management offers many corrective choices and it can even be used to target SQL Server 2000 and SQL Server 2005 servers.

Of course, you should always be using Windows Update to ensure that new security patches and service packs are downloaded for you to install at an appropriate time—this is the best way to stay up to date with the latest updates. Installing these without taking downtime is beyond the scope of this article, but some ideas were presented in the [December 2006 SQL Q&A column.](https://technet.microsoft.com/magazine/2006.12.sqlqa.aspx)

If you're trying to find general Microsoft resources on security, there are many destinations that will show up in your favorite search engine. To save you some time clicking around, I'll suggest two key white papers you should read.

"[SQL Server 2005 Security Best Practices–Operational and Administrative Tasks](https://technet.microsoft.com/en-us/sqlserver/bb331769.aspx)" and "[SQL Server 2008: Security Overview for Database Administrators](http://microsoft.com/sqlserver/2008/en/us/wp-sql-2008-security.aspx)."

For SQL Server 2005, the entry point into the security section of Books Online is the topic "[Security Considerations for Databases and Database Applications](https://technet.microsoft.com/library/ms187648(sql.90).aspx)." For SQL Server 2008, the equivalent Books Online entry point is the "[Security and Protection (Database Engine)](https://technet.microsoft.com/en-us/library/bb510589.aspx)."

As far as takeaways from this article are concerned, I want you to realize that there are some steps you need to go through to ensure the data you are storing in SQL Server is as secure as you need it to be. This is especially important when you inherit a SQL Server instance that someone else has been managing. It's just like buying a house from someone—you need to ask if the alarm works, if the yard is fenced in, and who has copies of the keys. Running through the list I've given in this article is a good start, but make sure you dig deeper in areas that are relevant to you. And as always, if you have any feedback or questions, drop me a line at [Paul@SQLskills.com](mailto:Paul@SQLskills.com).