http://www.sqlservercentral.com/articles/SQL+Server+2005+-+Security/sqlserver2005logi
Printed 2008/04/12 05:48PM

SQL Server 2005 Logins

By Brian Kelley, 2006/06/22

This is the first in a series of articles on SQL Server 2005 Security. We'll start right at the beginning inside SQL Server with logins. Logins are what SQL Server uses to determine if a given person/application has the right to connect to SQL Server. If you are familiar with any of the versions of SQL Server prior to 2005, the concept of logins hasn't changed, though new login types have been added.

Types of Logins

SQL Server 2005 maintains the 3 types of logins from earlier versions of SQL Server. It also adds logins mapped to certificates and asymmetric keys. A breakdown of the login types are:

Login types from earlier editions of SQL Server:

- SQL Server login
- Windows user login
- Windows group login

New login types for SQL Server 2005:

- Login mapped to a certificate
- Login mapped to an asymmetric key

SQL Server 2005's support of encryption within SQL Server itself allows for the support of the two new login options. However, the original 3 logins, corresponding to SQL Server logins and the two Windows-based ones, are and will likely continue to be the ones most commonly used.

SQL Server Logins

This is the traditional method to login to SQL Server. SQL Server stores both the username and the password (actually a hash of the password) with the *master*
database and verifies a login attempt to use a SQL Server login internally. Because some 3rd party products still do not use Windows authentication and clients from other operating systems may not be able to connect using Windows authentication, SQL Server logins have remained. Also, SQL Server logins allow connections from a Windows system in an untrusted domain or workgroup because in such cases Windows-based logins would fail (with a notable exception which is beyond the scope of this article).

While SQL Server logins are the same as they were in previous versions of SQL Server, the options we have with SQL Server logins have been increased with respect to password handling. One of the knocks against SQL Server logins was there were security weaknesses in how SQL Server handled the administration of passwords. The problems identified were:

- A member of the *sysadmin*
  fixed server role created a SQL Server login and set the password. However, there was no requirement for the user to change the password. Nothing within SQL Server forced this. Windows has a flag which indicates that a user must change the password on the next successful login, but SQL Server didn't have this. This is a security weakness. There is no reason for the administrator to know the user's password. SQL Server 2005 has an option, *MUST_CHANGE*, which forces the user to change the password upon first successful use of the login.
- There was no password complexity requirements on the password for a SQL Server login. About the only thing a DBA could check for was whether or not there was a blank password without resorting to

some password cracking tool. With SQL Server 2005 running on Windows Server 2003, the password complexity requirements set for the operating system (whether by local or Group Policy) can be applied to SQL Server logins now.
- There was no password expiration. Therefore, a password could stay in existence as long as the SQL Server did. A DBA could check the *updatedate* column in the syslogins system table to see when a password changed, however, this wasn't always a valid check. This column gets changed whenever anything about the login gets changed, to include the default database or language. Therefore, this column could reflect a change, even if it's unrelated to the password. Also, even if a DBA flags a particular login, there was no automated mechanism within SQL Server to disable the account. A DBA would have to execute the system stored procedure **sp_denylogin** manually to stop the login from being used. If installed on Windows Server 2003, SQL Server 2005 now can handle password expiration according to the requirements set by the operating system (again through the local security policy or Group Policy).
- There was no account lockout for entering too many bad passwords. Therefore, an attacker could use a brute force attack to attempt and crack the password. Nothing in SQL Server would disable the account. This is also remedied in SQL Server 2005 by checking the requirements set by the operating system.

All of these changes were made to enhance the security of SQL Server 2005. SQL Server 2005 doesn't require you to use any of these options. Furthermore, it's not an all or nothing choice. You can use these options on some logins and not on others. A case where you wouldn't use these options is when a 3rd party product has a hard-coded login and password. Obviously, the password complexity check and expiration policy does you no good because you wouldn't be able to change the password within the application. With that said, whenever you do have the choice to use the new password options, it is recommended that you do so.

Windows User and Group Logins

These methods of logging in to SQL Server remain unchanged from SQL Server 7.0 and 2000. Such user or group accounts must either come from the system the SQL Server is installed on or a trusted domain account. There aren't any new options in SQL Server 2005 with respect to Windows-based logins as security is already handled on the operating system side. Keep in mind that a Windows group is just that: a group. As a result, granting access to a Windows group grants access to any Windows user logins which are a member of that Windows group. For instance, the BUILTIN\Administrators group is granted access to SQL Server 2005 when you first install it. This corresponds to the local Administrators group on the system which SQL Server 2005 is installed. Any Windows accounts which are a member of this Administrators group can login to SQL Server.

Keep in mind that with Windows security groups, nesting of security groups is possible. In Windows NT 4.0 this nesting was limited. You could nest a global group inside a local group, but that was it. With Active Directory, however, there are several permutations of how groups can be nested and nesting can be many levels deep (though this is generally frowned upon because of the complexity it generates). As a result, when dealing with Windows groups you need to be aware of how many levels these groups may be nested and what the membership "tree" is from the group you grant access to your SQL Server.

While these logins haven't changed, SQL Server 2005 offers the ability to set the default database and language at the time the login is created. This differs from SQL Server 7.0 and 2000 where the system stored procedure **sp_grantlogin** only had one parameter, *@loginame*, which was used to grant the login access to SQL Server. If we wanted to set a different default database other than master and/or a different default language, that required the execution of the **sp_defaultdb** and **sp_defaultlanguage** system stored procedures respectively. In SQL Server 2005 there is no longer the need to execute those additional stored procedures. This will be discussed more under **Creating Logins**.

Certificates and Asymmetric Keys

If a certificate or asymmetric key is stored within SQL Server 2005, it can be used as a login. SQL Server 2005 isn't the first such application to allow logins using these sources. There have been 3rd party products which allowed the use of a smart card to log on to Windows and other operating systems for years now. An example of this would be the US Military's Defense Message System which used the US National Security Agency's Fortezza card. In addition, some products, such as HP's System Insight Manager, tends to use

certificates for connections between systems. Certificates and asymmetric keys aren't used very heavily as logins as of yet with respect to SQL Server 2005 and as this is a "basics" article, I'll postpone discussion of their use as logins to a later article.

New Syntax for Managing Logins

Given the new options for logins in SQL Server 2005, changes to the syntax for managing logins was necessary. Previously, the following stored procedures were used:

- **sp_addlogin**
- **sp_denylogin**
- **sp_droplogin**
- **sp_grantlogin**

One way to handle the new features would be to expand these stored procedures. However, Microsoft took a different approach. If logins are though of as objects within SQL Server just as tables, views, stored procedures, functions, etc. are, then there should be specific T-SQL commands to handle their management. As a result, there is now CREATE LOGIN, ALTER LOGIN, and DROP LOGIN.

Do the old stored procedures still work? Yes, they do, and their syntax is the same as it was in previous versions of SQL Server. The stored procedure sp_addlogin does not support any of the new options with respect to passwords in SQL Server 2005, however. As a result, if you are supporting SQL Server 2005 or developing programs for it which will deal with logins, it is best to use the new syntax.

Creating Logins

SQL Server 2005's new syntax for creating logins is:

> CREATE LOGIN *name* { WITH *options* | FROM *source* }

SQL Server Logins

The *options*
are for dealing with SQL Server logins and they correspond to some of the parameters available in the old **sp_addlogin**
system stored procedure (default database, default language, and SID) as well as new parameters based on the enhancements to password management. The options are:

> PASSWORD = '*password*' [HASHED] [MUST_CHANGE] [, *additional options* [, ...]]

When creating a SQL Server login, the *PASSWORD* must be set. A blank password can be given by specifying anything between the single quotes, but obviously, this is recommended against from a security perspective. When specifying the password, two optional arguments may be specified:

- *HASHED*
  tells SQL Server what is being specified is already in the form of a hash (the password is already "encrypted"). This is the same as specifying *'skip_encryption*' as the value of the *@encryptopt* parameter for **sp_addlogin**.
- *MUST_CHANGE*
  will tell SQL Server to prompt the user to change the password on the first successful login. However, if you choose this option you **must**
  also choose to turn on policy checking and password expiration (more on those later in the article). Also, this option is only supported on Windows Server 2003. If you attempt to choose this option on another operating system (Windows 2000 or XP), you'll receive the following error:

Msg 15195, Level 16, State 1, Line 1
The MUST_CHANGE option is not supported by this version of Microsoft Windows.

In order to turn on policy checking and password expiration, additional options must be specified. Those additional options are:

- SID = *SID*
- DEFAULT_DATABASE = *default database*
- DEFAULT_LANGUAGE = *default language*
- CHECK_EXPIRATION = { ON | OFF}
- CHECK_POLICY = { ON | OFF}
- CREDENTIAL = *credential name*

SID does the same thing as the *@sid* parameter for **sp_addlogin**. When creating the SQL Server login, the Security IDentification number (SID) is generated automatically unless you manually specify this additional option. Since the SID at the login level corresponds to the SID at the user level within a given database, there are times when you want to specify the SID to prevent orphaned database users. For instance, if you were extracting the logins from one SQL Server instance to create on another because you were transferring databases, you would want to specify the SID. This would eliminate having to execute the **sp_change_users_login** system stored procedure for each SQL Server login being transferred as the SIDs would stay in synchronization at the server level and the database level.

DEFAULT_DATABASE and *DEFAULT_LANGUAGE* are self-explanatory. If you don't specify these, SQL Server will set the login's default database to master and the default language to the current default language of the server.

CHECK_EXPIRATION and *CHECK_POLICY* tell SQL Server to enforce the settings on password found in the computer's effective local security policy. Since Group Policy overrides the local security policy, the effective setting may actually be in a Group Policy. When *CHECK_POLICY* is on, SQL Server 2005 will get password policies and enforce them. However, *CHECK_EXPIRATION* can still be turned off, even if you want to ensure password complexity, password history, and account lockout settings are observed and enforced. If you set *CHECK_POLICY* on, though, *CHECK_EXPIRATION* will also be on unless you explicitly turn it off. *CHECK_POLICY* is only fully enforced in Windows Server 2003. It can be turned on in Windows 2000, but only the password complexity is checked. Even with that said, the password complexity check in Windows 2000 just verifies the password isn't any of the following: null or empty, the name of the computer, the name of the login, 'password', 'admin', 'administrator', 'sa', or 'sysadmin'.

CREDENTIAL is an option which associates the login with what SQL Server 2005 calls a credential. A credential contains the authentication information (such as username and password) to connect to a resource outside of SQL Server. Since this is a basic article on logins, I won't go into any more detail on credentials.

Putting this all together, an example *CREATE LOGIN* command for a SQL Server login would be:

```
CREATE LOGIN TestLogin
WITH PASSWORD = 'Ch4ng3M3!' MUST_CHANGE,
DEFAULT_DATABASE = AdventureWorks,
CHECK_EXPIRATION = ON,
CHECK_POLICY = ON
```

Windows Logins

The FROM *source* is for Windows-based logins, certificates, and asymmetric keys. However, I'll only cover Windows-based logins in this article. Given that, the syntax is:

```
FROM WINDOWS [WITH Windows options [, ...] ]
```

FROM WINDOWS covers both Windows user accounts and Windows security groups. The *Windows options*

are:

- DEFAULT_DATABASE = *default database*
- DEFAULT_LANGUAGE = *default language*

Again, *DEFAULT_DATABASE* and *DEFAULT_LANGUAGE* are self-explanatory. However, the ability to set these two options at the same time the login is created is new. Such functionality did not and does not exist with **sp_grantlogin**.

An example *CREATE LOGIN* command for a Windows account would be:

```
CREATE LOGIN [BUILTIN\Users]
FROM WINDOWS
WITH DEFAULT_DATABASE = AdventureWorks
```

Deleting Logins

Getting rid of a login is extremely easy:

```
DROP LOGIN name
```

A word of caution with this command, however. SQL Server 2005 will allow you to drop the login even if the login has been mapped into one or more databases as a user. Therefore, be sure to verify the login does not exist as a user in all databases before dropping the login.

Modifying Logins

There are two ways to modify logins. The first is to enable or disable the login. The second is to make changes to the login's properties. Both ways begin with *ALTER LOGIN*:

ALTER LOGIN *name* { *status* | WITH *option* [, ...] }

Enabling and Disabling Logins

A login may be set to one of two statuses:

- ENABLE
- DISABLE

ENABLE means the login can be used to connect to SQL Server. *DISABLE* toggles the login so it cannot be used to connect. To disable the login TestUser, we'd execute the following command:

```
ALTER LOGIN TestLogin DISABLE
```

Notice that I've disabled a SQL Server login. This represents new functionality in SQL Server 2005. Previously, I could deny a Windows login from connecting by executing the sp_denylogin system stored procedure. However, there was no way to temporarily prevent a SQL Server login from connecting. With SQL Server 2005, any login can be disabled. This is perfect for situations where a given application connects using a SQL Server login and you need to perform some sort of database maintenance. The login can be disabled until your maintenance is complete.

Setting Options

There are several options which can be executed using the ALTER LOGIN statement. All of these apply to the *WITH option [, ...]* portion of the ALTER LOGIN. They are:

- Resetting the password on the login
- Setting the default database
- Setting the default language

- Changing the login name itself (renaming the login)
- Setting whether or not to check the password policy
- Setting whether or not to check password expiration
- Setting a credential for the login (or unsetting a credential)

These options can be stacked together. Let's look at each of them in turn, with the exception of credentials.

Resetting the Password

The password options are:

> PASSWORD = '*new password*' [OLD_PASSWORD = '*old password*' | *secadmin password option* [*secadmin password option*]

The two secadmin password options are: MUST_CHANGE and UNLOCK. The first forces the user to change the password upon first login. The second option unlocks a login which has been locked due to too many failed login attempts. If MUST_CHANGE is set, password policy and expiration must also be set (see below). An example of changing the password on a locked account is:

```
ALTER LOGIN TestLogin WITH PASSWORD = 'MyNewP4ssw0rd!' UNLOCK
```

Changing the Default Database or Language

Changing the default database and language are similar:

- DEFAULT_DATABASE = *database*
- DEFAULT_LANGUAGE = *language*

An example where both options are set (and an example of stacking options is):

```
ALTER LOGIN TestLogin
WITH DEFAULT_DATABASE = master,
DEFAULT_LANGUAGE = us_english
```

Renaming the Login

Renaming the login is new to SQL Server 2005. The syntax is the following:

> NAME = *new login*

Here we can rename TestLogin to TestNewLogin:

```
ALTER LOGIN TestLogin WITH name = TestNewLogin
```

Checking Password Policy and Expiration

The settings to check password policy and expiration are:

- CHECK_POLICY = { ON | OFF }
- CHECK_EXPIRATION = { ON | OFF }

If CHECK_EXPIRATION is set to ON, CHECK_POLICY must also be set to ON. Otherwise, the following error will be returned:

```
Msg 15122, Level 16, State 1, Line 1
The CHECK_EXPIRATION option cannot be used when CHECK_POLICY is OFF.
```

Putting this together with a password reset we could execute the following:

```
ALTER LOGIN TestLogin
```

```
WITH PASSWORD = 'MyNewP4ssw0rd!' MUST_CHANGE,
CHECK_POLICY = ON,
CHECK_EXPIRATION = ON
```

Concluding Remarks

SQL Server 2005 maintains the logins we are familiar with from SQL Server 7.0 and 2000, however, it also allows logins for certificates and asymmetric keys. For the most part, however, Windows-based logins (users and groups) as well as SQL Server logins are what will be used by our applications. Therefore, if you're familiar with logins from the earlier versions of SQL Server, you've got a good start on logins in SQL Server 2005.

What is different with the new version is the syntax for handling logins. SQL Server 2005 introduces CREATE, DROP, and ALTER LOGIN statements which provide greater functionality than the system stored procedures from SQL Server 7.0 and 2000. I've tried to cover these three commands and their options, with the exception of when the commands deal with certificates, asymmetric keys, and credentials (which will be covered in a later article). These commands may take some getting used to as their syntax differs drastically from anything we might find in the earlier versions. With that said, the system stored procedures for handling logins from SQL Server 7.0 and 2000 are still available for backward compatibility. They won't allow access to SQL Server 2005 specific features, such as the ability to apply the computer's security policy with respect to passwords. To take advantage of these features, the new commands must be used.

© 2006 by K. Brian Kelley. Author of *Start to Finish Guide to SQL Server Performance Monitoring*.
| Professional Site | Database Blog | Infrastructure Architecture Blog |