There are a variety of commercial security tools available to audit Microsoft (MS) SQL Server databases. However, there can be instances when their application is not practical, including:

- For smaller companies in which the cost may be prohibitive
- For larger holding companies or geographically dispersed companies that do not have full network connectivity between the centre and its subsidiaries
- For consultancies performing external reviews that are not given permission to install or run tools that require full database administrator privileges and, hence, the administrator password. Furthermore, the audited entity has no oversight of what the tool does or what effect it is likely to have on mission-critical databases.

The following approach to auditing MS SQL Server databases using computer-assisted audit techniques (CAATs) in conjunction with information taken directly from the MS SQL Server database offers a solution to the issues identified. It describes a cost-effective solution for auditing MS SQL Server databases to a company's own standards using tools already available in the enterprise.

SQL SERVER DATABASES

All MS SQL Server installations contain information about the installed databases. To retrieve the information, one must understand how it is stored and the tools available to extract it.

- **SQL server catalog views**—MS SQL Server databases contain information about data known as catalog metadata. This provides details about the database's configuration options, users, objects and more. Catalogue metadata are accessed through catalog views. Catalogue views provide an organised view of the catalog metadata. The views are grouped into categories, one of which is security. A full list of the catalog views along with more detailed explanations may be found on Microsoft's Developer Network.

- **SQL Server Management Studio**—This is a tool for accessing, configuring, managing, administering and developing SQL server databases. It also allows users to query the database using SQL, formatting the output as desired and writing the results to file (if required).

- **SQL server structure**—Each instance of the MS SQL Server has four system databases (master, model, tempdb and msdb) and one or more user databases. Security is built around the master/server level (server roles) and the user database level (database roles).

**OUTPUTTING SQL SERVER CATALOG VIEWS**

Figure 1 offers an example of an MS SQL Server script (a full script can be downloaded from the ISACA Knowledge Center). Once generated, these scripts can be transferred to the database administrator to be run over the required database(s). One comma-separated values (CSV) file is produced for each server-level view. A CSV file is also produced for each database-level view, but it contains separate output for all installed databases (the required setup, formatting and configuration options are described in figure 2).

**ANALYSING SQL SERVER CATALOG VIEWS**

The files can then be imported into CAATs tools for analysis and comparison. Examples of MS SQL Server catalog views include:

- **Configurations**—The entity being audited should have a policy on how its MS SQL Server databases are configured. Much of the configuration is reflected in the MS SQL Server parameters, which can be retrieved from the sys.configurations catalog view. Examples of parameters include those for enabling the command shell or allowing client applications on remote computers to use an administrator connection (a full list of SQL server configuration options exists on Microsoft’s Developer Network).
Figure 1—SQL Script Output to CSV Text Files

-- Be sure you are in SQL Command Mode
-- Replace the output destination "C:\SQLAUDIT\" as required

set nocount on

-- (a) SQL Server Configuration, output to SYS_Configuration.txt
:OUT C:\SQLAUDIT\SYS_Configuration.TXT
SELECT * FROM SYS.CONFIGURATIONS
GO

-- (b) Authenticated Users / roles, output to SYS_SERVER_PRINCIPALS.txt
:OUT C:\SQLAUDIT\SYS_Server_principals.TXT
SELECT * FROM SYS.SERVER_PRINCIPALS
GO

-- (c) SQL Logins, output to SYS_SQL_LOGINS.txt
:OUT C:\SQLAUDIT\SYS_SQL_Logins.TXT
SELECT * FROM SYS.SQL_LOGINS
GO

-- (d) Server Role Members (links with principals above), output to SYS_SERVER_ROLE_MEMBERS.txt
:OUT C:\SQLAUDIT\SYS_Server_role_members.TXT
SELECT a.role_principal_id, b.name as role_principal_name, a.member_principal_id, c.name as member_principal_name
FROM (sys.server_role_members a INNER JOIN sys.server_principals b ON a.role_principal_id = b.principal_id)
INNER JOIN sys.server_principals c ON a.member_principal_id = c.principal_id;
GO

-- All other databases ------------------------------

-- (e) Database principals, output to DATABASE_PRINCIPALS.txt
:OUT C:\SQLAUDIT\DATABASE_PRINCIPALS.TXT
EXEC sp_MSforeachdb 'USE ? SELECT ''?'' as database_name, * FROM SYS.DATABASE_PRINCIPALS'
GO

-- (f) Database Role Members (links with principals above), output to MASTER_DATABASE_ROLE_MEMBERS.txt
:OUT C:\SQLAUDIT\DATABASE_ROLE_MEMBERS.TXT
EXEC sp_MSforeachdb 'USE ? SELECT ''?'' as database_name, a.role_principal_id, b.name as role_principal_name,
a.memberPrincipal_id, c.name
FROM (sys.database_role_members a INNER JOIN sys.database_principals b ON a.role_principal_id = b.principal_id)
INNER JOIN sys.database_principals c ON a.memberPrincipal_id = c.principal_id'
GO

Source: Ian Cooke. Reprinted with permission.

Figure 2—Formatting and Configuration Requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL server command mode</td>
<td>By using the Database Engine Query Editor in MS SQL Server Management Studio, one can write and edit queries as SQLCMD scripts. Use SQLCMD scripts when it is necessary to process Windows system commands and Transact-SQL statements in the same script. Here it is used to output to selected file(s). Ensure that SQLCMD Mode is selected.</td>
</tr>
<tr>
<td>Output to text file</td>
<td>Right click on the SQLCMD line (:OUT) in SQLCMD mode and choose query options, 'select text'. Select output format 'comma delimited', and check 'include column headers in the result set' (see Figure 3).</td>
</tr>
<tr>
<td>sp_MSForEachDB</td>
<td>The undocumented stored procedure ‘For Each Database’ comes with the MS SQL Server and can be found in the master database. It is used to process a single SQL query against all installed databases. In this instance, a query over a database catalog view is run over all databases.</td>
</tr>
</tbody>
</table>

Source: Ian Cooke. Reprinted with permission.
As noted, this catalogue view can be output to a CSV file. A sample output from the configuration’s CSV file can be seen in figure 3. The first line shows the MS SQL Server field names. The field name relates to the defined layout for the configurations catalogue view. The layout for all SQL server catalogue views are available on Microsoft’s Developer Network.

All common CAATs tools (as well as Microsoft Excel and Microsoft Access) allow for the importation of CSV files. Once the configurations are imported to the CAATs tool, they can be analysed for compliance to the entity’s standards. These standards could be based upon any source of assurance for an MS SQL Server database, e.g., the Center for Internet Security Benchmarks, or a document developed by the entity. (ISACA’s Microsoft SQL Server Database Audit/Assurance Program does not use catalogue views, although the items could be tested by referencing them.)

Once one database is analysed and known to be compliant to required standards, it can be used as a ‘master configuration’ using the CAATs tool to compare the configurations of interest across all of the organisation’s MS SQL Server databases. This can be done by joining the field configuration_id displaying all records where the value is not equal. In this manner, noncompliant configurations are quickly flagged for follow-up and review.

If required, the review of the configurations can be repeated periodically. This would allow changes to be tracked and used as part of a continuous monitoring audit programme.

- **Server principals**—An MS SQL Server provides two methods of authenticating to the database: Windows authentication or mixed mode. Windows authentication uses MS Windows security to validate all of the database accounts and passwords against the Windows operating system. It is the mode recommended by Microsoft. Mixed mode allows for both Windows authentication and MS SQL Server authentication. With MS SQL Server authentication, an explicit user account and password are required to access the database.

The entity being audited should have a policy on how these passwords are configured. For Windows authentication, these will be as per the Windows server. From Windows Server 2003 onwards, mixed mode can be configured to validate the password against the Windows server. In a mixed mode environment, the mode in use for a given user can be seen in the sys.server_principals catalogue view (figure 1[b]). The field ‘type’ will be ‘S’ for an SQL login or ‘U’ for a Windows login.

From SQL Server 2005 onwards, the password hashes for SQL login users are stored in the view sys.sql_logins catalogue view. To audit the passwords, one must request that the MS SQL Server database administrator output the contents of the view (figure 1[c]) to a CSV file, as discussed. Once one has the password hashes, these can be validated by running them through a password cracking tool, e.g., Hashcat.

In an MS SQL Server, anything that can be granted a right to perform an activity is called a principal. So, fundamentally, principals include logins, users and roles, for example. Principals can also be separated into server principals and database principals.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow updates</td>
<td>Allow updates to system tables</td>
</tr>
<tr>
<td>remote access</td>
<td>Allow remote access</td>
</tr>
<tr>
<td>remote admin connections</td>
<td>Dedicated Admin Connections are allowed from remote clients</td>
</tr>
<tr>
<td>Agent XPs</td>
<td>Enable or disable Agent XPs</td>
</tr>
<tr>
<td>SQL Mail XPs</td>
<td>Enable or disable SQL Mail XPs</td>
</tr>
<tr>
<td>Database Mail XPs</td>
<td>Enable or disable Database Mail XPs</td>
</tr>
<tr>
<td>SMO and DMO XPs</td>
<td>Enable or disable SMO and DMO XPs</td>
</tr>
<tr>
<td>Ole Automation Procedures</td>
<td>Enable or disable Ole Automation Procedures</td>
</tr>
<tr>
<td>Web Assistant Procedures</td>
<td>Enable or disable Web Assistant Procedures</td>
</tr>
<tr>
<td>xp_cmdshell</td>
<td>Enable or disable command shell</td>
</tr>
<tr>
<td>Ad Hoc Distributed Queries</td>
<td>Enable or disable Ad Hoc Distributed Queries</td>
</tr>
<tr>
<td>Replication XPs</td>
<td>Enable or disable Replication XPs</td>
</tr>
</tbody>
</table>

Source: Ian Cooke. Reprinted with permission.
Permissions are what allow principals to perform activities in an MS SQL Server. Depending on the scope, the permission granted is either a server permission or a database permission.

Roles are used to group together permissions or other roles. They are a means of facilitating the granting of multiple permissions or roles to users. An MS SQL Server provides some predefined roles to help in database administration. Fixed server roles are defined at the server level and have server-level permissions. They cannot be added, removed or changed. Each member of a fixed server role can add other users to the role. Since MS SQL Server 2012, one can also create user-defined server roles.

Fixed server roles can also be seen in the sys.server_principals view (the field ‘type’ is ‘R’), while the users who have been allocated the fixed server roles can be seen in the sys.server_role_members view (figure 1[d]). These should be reviewed for appropriateness and separation of duties.

- **Database principals**—Fixed database roles are defined at the database level and exist in each database. Members of the db_owner and db_securityadmin database roles can manage fixed database role membership; however, only members of the db_owner database role can add members to the db_owner fixed database role.

User-defined (or flexible) database roles allow one to create one’s own roles. After a role is created, one can configure the database permissions of the role by using grant, deny and revoke, for example.

Using the sys.database_principals view via the sp_MSforeachdb stored procedure (figure 1[e]), one can obtain a list of the database roles in use for each database. The users allocated these roles can be retrieved from sys.database_role_members (figure 1[f]). These should also be reviewed for appropriateness and separation of duties.

As there are many MS SQL Server catalog views, it is not possible to discuss them all in this article. Those mentioned previously are for illustration purposes. The views used depend on the purpose of the audit, but could also include:

- **Sys.databases**, which lists all the installed databases. This could be used to ensure that all databases are required, in use, and secured, and that there is a separation between test and live databases.

- **Sys.server_permissions**, which lists all the permissions that have been defined at the server level. This could be reviewed to ensure that the server permissions are appropriate.

- **Database_permissions**, which lists all the permissions that have been defined at the database level. This could be reviewed to ensure that the database permissions are appropriate.

- **Database_objects**, which lists all the objects for a given database. This could be cross-checked with permissions to ensure that the database permissions are appropriate.

A full list of MS SQL Server catalog views, including those for auditing and encryption, along with more detailed explanations can be found on Microsoft’s Developer Network.

The key point is that any of the views may be output to CSV files and imported into a CAATs tool. These may then be:

- Compared to other SQL server databases, including a master that one knows to be compliant to one’s standards
- Compared to development, test or QA versions of the production database (useful for change control)
- Compared to other sources of data, including data from internal or external entities

**BENEFITS OF CAATS FOR SQL SERVER DATABASES**

Benefits of the proposed CAATs solution for auditing MS SQL Server databases include cost, as many organisations are already using CAATs software. The approach also allows for total transparency as the database administrator of the audited entity can review the SQL that is being run over the database to ensure that it will have no adverse effects on the production environment. Furthermore, the approach allows external consultants and geographically dispersed companies to request that the queries are run by the local database administrator without the need to compromise security (the administrator password) or install any additional software. Once the queries have been run,
they can be securely transferred for analysis. Query results can be compared against other databases from the audited entity or known compliant (master) databases to highlight areas of audit concern. Query results can also be compared against preproduction databases and other sources of data, such as the company payroll. Finally, the entire process can be repeated, reconfigured (if required) and used as part of a continuous monitoring and/or audit.

ENDNOTES

8 Op cit, Microsoft Developer Network, Catalog Views (Transact-SQL)
9 ISACA, Knowledge Center, ‘Sources of Assurance for a SQL Server Database’, www.isaca.org/Groups/Professional-English/sql-servers/GroupDocuments/Sources_of_Assurance_for_a_SQL_Server_Database_Update_2.pdf
16 Hashcat.net, http://hashcat.net/hashcat/
21 Microsoft Developer Network, ‘Permissions of Fixed Database Roles (Database Engine)’, http://msdn.microsoft.com/en-us/library/ms189612(v=sql.100)
23 Op cit, Microsoft Developer Network, ‘Catalog Views (Transact-SQL)’