6 SQL Server Security Basics Every Database Admin Should Know

Too many data breaches are caused by poorly secured database servers. Microsoft SQL Server is a popular enterprise solution, but it is also complex to understand and configure. Here are 6 SQL Server security features that you should know to keep your server data protected.

1. SQL Server Authentication vs. Windows Authentication

Microsoft SQL Server supports two authentication options:

- **Windows Authentication** relies on Active Directory (AD) to authenticate users before they connect to SQL It is the recommended authentication mode because AD is the best way to manage password policies and user and group access to applications in your organization.
- **SQL Server Authentication** works by storing usernames and passwords on the database server. It can be used in situations where Active Directory is not available.
  You can use SQL Server and Windows Authentication at the same time (mixed-mode), but, whenever possible, use Windows Authentication exclusively.

If you must use SQL Server Authentication, make sure that the default sa account is either disabled or has a strong password that is changed frequently, because this account is often targeted by hackers. SQL Server accounts, including sa (if it's enabled), can be managed using the SQL Server Management Studio service or the ALTER LOGIN Transact-SQL (T-SQL) command.

2. Server Logins and Roles

Regardless of the authentication method you choose, there are two types of login that you configure to SQL instances: user logins and server logins. I'll discuss user logins in the next section. Server logins let users establish a connection to a SQL Server instance. Each server login is assigned one or more *server roles* that enables it to perform specific actions on the instance. By default, server logins are assigned the *public* server role, which gives basic access to the instance. Other available roles include *bulkadmin*, *sercurityadmin*, *dbcreator* and *serveradmin*.

Server logins can be created using T-SQL or the SQL Server Management Studio. When creating a server login, you must specify a default database. Server logins are associated with a user login in the default database. It's worth noting that a server login name and the name of its associated user login don't need to match. If there is no associated user object in the default database, the server login will be denied access unless the server role assigned to the login has access to all databases. Server logins can be mapped to a user in one or more databases, and you can create users when setting up server logins.

3. Database Users, Schema and Roles

When creating a user login, you need to specify the database it will be associated with, a username and a default schema that will be applied to all objects that the user creates if no other schema is specified. SQL Server schemas are collections of objects, like tables and views, logically separated from other database objects, which makes it easier to manage access and means there is no need to use the schema name when running T-SQL commands against a database.

The default schema for user-defined objects is *dbo*. The other default schema is *sys*; it owns all system objects.

In the same way that server logins are assigned server roles, user logins are assigned database roles, which assign rights in databases. Server database roles include *public*, *db accessadmin*, *db owner* and *db securityadmin*.

4. Securables and Permissions

When server or database roles would give a user too much or too little access, you can assign one or more securables instead. Securables exist at the server, schema and database levels; they are SQL Server resources that can be accessed by server and user logins. For example, using securables, you could give a server login access to a specific table or function only, a level of granularity that isn't possible by assigning a role to a login.

Permissions are used to grant access to SQL Server securables. You might grant permission to just view data or just to modify data. The GRANT, DENY and REVOKE T-SQL statements are used for configuring permissions.

However, permissions can be complicated. For example, setting DENY permissions on a securable prevents permission inheritance on lower-level objects. But the column-level GRANT permission overrides DENY at the object level, so DENY permission set on a table is overridden by GRANT permission on a column. Because permissions can be complex, it's always worth checking effective permissions using T-SQL. The following command determines JoeB's permissions granted on an object, in this case a table called 'employees'.

```
SELECT * FROM fn_my_permissions('joeb', employees);
GO
```

5. Data Encryption

SQL Server supports multiple encryption options:

- **Secure Sockets Layer (SSL)** encrypts traffic as it travels between the server instance and client application, much like internet traffic is secured between browser and server. Additionally, the client can validate the server's identity using the server's certificate.
- **Transparent Data Encryption (TDE)** encrypts data on disk. More specifically, it encrypts the entire data and log files. Client applications don't need to be changed when TDE is enable
- **Backup Encryption** is similar to TDE but encrypts SQL backups instead of the active data and log files.
- **Column/Cell-Level Encryption** makes sure that specific data is encrypted in the database and remains so even when it is stored in memory. Data is decrypted using a function and requires changes to client applications to
- **Always Encrypted** is an improvement on Column/Cell-Level Encryption in that it doesn't require any changes to client applications; data stays encrypted over the network, in memory and on disk. It also protects sensitive data from the prying eyes of privileged SQL Server users. But you can encounter some issues with this encryption option — because SQL Server can't read the data, some indexing and functions won't work.

6. Row-Level Security

Row-Level Security (RLS) allows organizations to control who can see rows in a database. For example, you could restrict users to seeing only rows that contain information about their clients.

RLS consists of three main parts: a predicate function, a security predicate and a security policy. The predicate function checks whether the user executing the database query can access a row based on logic. For instance, you could check if the username of the user running the query matches a field in one of the row's columns. A predicate function and security predicate are defined together in a function to either silently filter the results of a query without raising errors or to block with an error if row access is denied. Finally, a security policy binds the function to a table.