CHAPTER 6

Designing Security for Applications

Chapters 1 through 4 concentrated on securing SQL Server itself both in terms of who can log into the server and in terms of what a user can do with the data in the databases. It is very unusual, however, for users to work directly with a database server; the typical scenario involves a client working with an application that, in turn, issues queries to the database server. This concept of a client-server relationship is a very old one that dates back to the early days of computers, and even application designs that have several layers between the client and the database server are just combinations of multiple client-server relationships.

In this chapter, I will move away from focusing just on the database server and instead look at ways to secure the conversation between the client and server. I will first cover the options traditional application designers have for securing data as it travels between the layers in a multilayer application. Then I will cover the options both for securely authenticating a user’s identity and for keeping the data private as it travels over the Internet.

The final part of the chapter will look at the SQL Injection Attack, a widespread potential source of attack to guard against, which reinforces the importance of an integrated approach to security. This discussion is followed by a list of recommendations that will serve as a reminder of what you need to consider when creating your own integrated approach.

As you are reading, remember that once a user logs in, SQL Server takes over all responsibilities related to securing data. Any permissions or user rights granted outside SQL Server do not matter. In fact, the best place to start a discussion of application security is with a review of the options for authentication and authorization in SQL Server.

Authenticating Access to the Server

Your first decision in designing security for applications will be deciding how a user will authenticate with SQL Server. As you saw in Chapter 2, the version of the database server greatly affects this decision. All versions offer either SQL
Server authenticated or Windows authenticated logins, but because SQL Server authenticated logins are basically the same in all versions, let’s look at some options for logging in with SQL Server accounts.

**Obscuring SQL Server Logins**

No matter how a user authenticates, you have to recognize that if he can access a server directly, he can bypass the application that is supposed to manage the data. In most cases, it is not desirable to have users issuing queries and sending updates directly to the database. Most applications perform at least some validation on the data before sending changes to the database, and almost all applications transform data from one format to another. It is also very common for the application to control access to the data instead of setting database permissions.

For SQL Server authenticated logins, obscuring the account name and password is a fairly simple modification to existing programs. Consider using an algorithm that turns the user-supplied password into a string of characters the user cannot derive by knowing the original password. The technique involves adding characters both to the login account and to the password. By adding characters known only to the application programmers, you can obscure the SQL Server account name from the user. For example, if Joe Smith's login ID for the application were JoeS, you could add two Os to the beginning and a Z to the end to create his SQL Server account 00JoeSZ. The only way Joe could log into SQL Server directly would be if he knew the algorithm used to add characters to his application ID.

A variation on this technique is to use a **hashing algorithm** to transform Joe's application ID into a series of characters and numbers. Usually, this technique employs a hashing key that can be either hard-coded into the application, supplied during installation of the application, or created using the user's own password. The following example shows the account name “JoeS” hashed to a 32-character (i.e., a 16-byte) value:

```
MD5 Hash ID=JoeS 
SQL Server Login=7120f7788d4ca3f8ae8ac07e63703d6c
```

As you can see, it is highly unlikely a user will be able to guess the login account name. It is also the nature of the MD5 hashing algorithm that it will be very difficult for either the user or an attacker to discover the key used in the transformation; therefore, this technique complements strong passwords because it will be difficult, if not impossible, to determine valid account names to use to log in.