Encrypting Connection Strings in Web.config

by Wei-Meng Lee
02/15/2005

One of the best practices in ASP.NET is to save your database connection strings in the Web.config file instead of hard-coding it in your code. This allows you to change database servers easily, without needing to modify your code. As an additional protection, it is always better to use integrated Windows security to access your database, rather than using SQL Server authentication, and thus including your SQL server credentials in the connection string. Either way, it's not such a good idea to save your connection strings as plain text in Web.config -- you should ideally encrypt the connection strings so that it leaves no chance for a potential hacker to easily get more information about your database server.

In ASP.NET 2.0, Microsoft has taken this further by allowing you to encrypt the connection strings in Web.config, all without much plumbing on your part.

Using the `DataProtectionConfigurationProvider` and `RSAProtectedConfigurationProvider` for Encryption

To see how you can encrypt the connection strings stored in the Web.config file, you will configure a `GridView` control to bind to an `SqlDataSource` control. The connection string used by the `SqlDataSource` control is saved in the Web.config file. You then encrypt the connection strings, using the two Protection Configuration Providers available in .NET 2.0.

1. Launch Visual Studio 2005 and create a new Website project. Name the project as `C:\EncryptConfig`.

2. Populate the default form with a `GridView` control and configure it to use an `SqlDataSource` control. Configure the `SqlDataSource` control using the Choose Data Source drop-down list in the `GridView` Tasks menu (see Figure 1) to connect to the Authors table in the pubs database in SQL Server 2000. In particular, ensure that the connection string to the database is stored in Web.config.

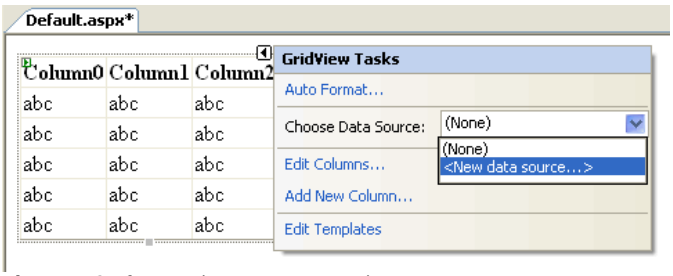


Figure 1: *Configuring the `GridView` control.*

The Source View of the `GridView` and `SqlDataSource` controls look like this:

```
<asp:GridView ID="GridView1" runat="server" DataKeyNames="au_id"
     DataSourceID="SqlDataSource1" AutoGenerateColumns="False">
   <Columns>
       <asp:BoundField ReadOnly="True" HeaderText="au_id" DataField="au_id"
            SortExpression="au_id"></asp:BoundField>
       <asp:BoundField HeaderText="au_lname" DataField="au_lname"
            SortExpression="au_lname"></asp:BoundField>
       <asp:BoundField HeaderText="au_fname" DataField="au_fname"
            SortExpression="au_fname"></asp:BoundField>
       <asp:BoundField HeaderText="phone" DataField="phone"
            SortExpression="phone"></asp:BoundField>
       <asp:BoundField HeaderText="address" DataField="address"
            SortExpression="address"></asp:BoundField>
       <asp:BoundField HeaderText="city" DataField="city"
            SortExpression="city"></asp:BoundField>
       <asp:BoundField HeaderText="state" DataField="state"
            SortExpression="state"></asp:BoundField>
       <asp:BoundField HeaderText="zip" DataField="zip"
            SortExpression="zip"></asp:BoundField>
       <asp:CheckBoxField HeaderText="contract" SortExpression="contract"
            DataField="contract"></asp:CheckBoxField>
   </Columns>
</asp:GridView>

<asp:SqlDataSource ID="SqlDataSource1" runat="server"
     SelectCommand="SELECT * FROM [authors]"
     ConnectionString="<%$ ConnectionStrings:pubsConnectionString %>">
</asp:SqlDataSource>
```

3. The default form should now look very much like Figure 2.

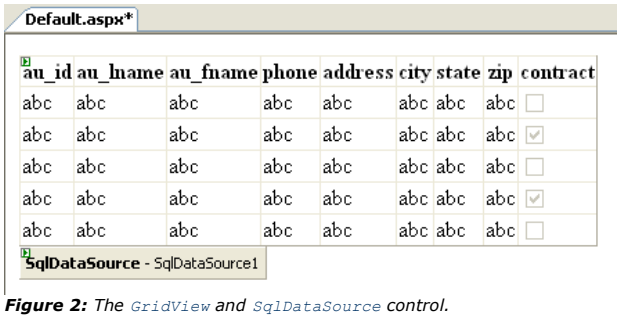


Figure 2: *The `GridView` and `SqlDataSource` control.*

4. The Web.config file will now contain the connection string:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
    <connectionStrings>
        <add name="pubsConnectionString"
             connectionString="Data Source="(local)";
                 Initial Catalog=pubs;Integrated Security=True"
             providerName="System.Data.SqlClient" />
    </connectionStrings>
    <system.web>
    ...
```

5. Switch to the code-behind of the default form and add in the `EncryptConnStr()` method. The `EncryptConnStr()` method first retrieves the Web.config file and then applies encryption to the specified section (<connectionStrings>, in this case) of the file using the Protection Configuration Provider indicated (passed in via the protectionProvider parameter).

```
Imports System.Configuration
Imports System.Web.Security

    Public Sub EncryptConnStr(ByVal protectionProvider As String)
        '---open the web.config file
        Dim config As Configuration = _
           ConfigurationManager.OpenWebConfiguration( _
           Request.ApplicationPath)
        '---indicate the section to protect
        Dim section As ConfigurationSection = _
           config.Sections("connectionStrings")
        '---specify the protection provider
        section.SectionInformation.ProtectSection(protectionProvider)
        '---Apple the protection and update
        config.Save()
    End Sub
```

6. Also, add in the `DecryptConnStr()` method. The `DecryptConnStr()` method will decrypt the encrypted connection strings in web.config:

```
    Public Sub DecryptConnStr()
        Dim config As Configuration = _
           ConfigurationManager.OpenWebConfiguration( _
           Request.ApplicationPath)
        Dim section As ConfigurationSection = _
           config.Sections("connectionStrings")
        section.SectionInformation.UnProtectSection()
        config.Save()
    End Sub
```

7. Note that the `UnProtectSection()` method, unlike `ProtectSection()`, does not require a provider name. When a section is encrypted, information regarding the provider that performed the encryption is stored in the Web.config file. UnProtectSection will use that information to determine which provider to use to decrypt the data.

8. There are two protection configuration providers available for your use -- `DataProtectionConfigurationProvider` and `RSAProtectedConfigurationProvider`. The `DataProtectionConfigurationProvider` uses the Windows DPAPI to perform encryption. The `RSAProtectedConfigurationProvider` uses the public-key algorithm available in the .NET Framework's `RSACryptoServiceProvider` class to perform encryption.

9. To test the `EncryptConnStr()` method, call it in the `Form_Load` event (you should only call the Encypt() method once) with the indicated Protection Configuration Provider:

```
    Protected Sub Page_Load(ByVal sender As Object, _
                            ByVal e As System.EventArgs) _
                            Handles Me.Load
        Encrypt("DataProtectionConfigurationProvider")
        '--or--
        ' Encrypt("RSAProtectedConfigurationProvider")
    End Sub
```

10. If you use the `DataProtectionConfigurationProvider` provider, your connection string will now look like this:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <protectedData>
      <protectedDataSections>
         <add name="connectionStrings" provider="DataProtectionConfigurationProvi
              inheritedByChildren="False" />
      </protectedDataSections>
   </protectedData>
   <appSettings/>
      <connectionStrings>
         <EncryptedData>
            <CipherData>
               <CipherValue>AQAAANCMnd8.............p4nZCszebgXs=</CipherValue>
            </CipherData>
         </EncryptedData>
      </connectionStrings>
   <system.web>
```

11. If you use the `RSAProtectedConfigurationProvider` provider, your connection string will now look like this:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
   <protectedData>
      <protectedDataSections>
         <add name="connectionStrings" provider="RSAProtectedConfigurationProvide
              inheritedByChildren="False" />
      </protectedDataSections>
   </protectedData>
   <appSettings/>
      <connectionStrings>
         <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
                        xmlns="http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod Algorithm=
                             "http://www.w3.org/2001/04/xmlenc#tripledes-cbc" /
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
               <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
                  <EncryptionMethod Algorithm=
                                          "http://www.w3.org/2001/04/xmlenc#rsa-1
                  <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                     <KeyName>RSA Key</KeyName>
                  </KeyInfo>
                  <CipherData>
                     <CipherValue>J7HkJ/9i...c2usLA=</CipherValue>
                  </CipherData>
               </EncryptedKey>
            </KeyInfo>
            <CipherData>
               <CipherValue>k6w9KUZ...5p2AP5gQ==</CipherValue>
            </CipherData>
         </EncryptedData>
      </connectionStrings>
   <system.web>
```

12. Notice that the <protectedData> section added to Web.config contains information needed to decrypt the connection strings. More importantly, <protectedData> doesn't contain the decryption key. For example, if you use the Windows `DataProtectionConfigurationProvider`, the decryption key is auto-generated and saved in the Windows Local Security Authority (LSA).

13. The really cool thing about encrypting the Web.config file is that the process of decrypting the required connection string is totally transparent to the developer. Controls and code that need to access the connection string will automatically know how to encrypt the encrypted information. However, if you want to decrypt the Web.config file so that you can make modifications to it, simply call the `DecryptConnStr()` method.

You can check if a section is protected by using the IsProtected property, like this:

```
If Not section.SectionInformation.IsProtected Then
   section.SectionInformation.ProtectSection(protectionProvider)
   config.Save()
End If
```

You can encrypt almost all the sections in Web.config, with the exceptions of some sections such as <httpRuntime> and <processModel>. This is because these sections are accessed by parts of the unmanaged code in ASP.NET.

Programmatically Add a New Connection String

You can programmatically add a new connection string to the Web.config file, even though it is encrypted. The following AddConnString() method adds a new connection string named NorthWindConnectionString to an encrypted Web.config:

```
   Public Sub AddConnString()
        '---add a connection string to Web.config
        Dim config As Configuration = _
               ConfigurationManager.OpenWebConfiguration( _
               Request.ApplicationPath)
        config.ConnectionStrings.ConnectionStrings.Add _
           (New ConnectionStringSettings("NorthwindConnectionString", _
           "server=localhost;database=northwind;integrated security=true"))
        config.Save()
    End Sub
```

If you were to decrypt the Web.config file, you will find the newly added connection string:

```
<connectionStrings>
   <add name="pubsConnectionString" connectionString="Data Source="
              (local)";Initial Catalog=pubs;Integrated Security=True"
        providerName="System.Data.SqlClient" />

   <add name="NorthwindConnectionString" connectionString="server=localhost;
              database=northwind;integrated security=true" />
</connectionStrings>
```

Summary

In this article, I have discussed how easy it is to encrypt connection strings in the Web.config file using the two Protection Configuration Providers -- `DataProtectionConfigurationProvider` and `RSAProtectedConfigurationProvider` -- for encryption. Since it is now so easy to perform encryption in Web.config, there is really no reason for not doing so.

Additional Resources

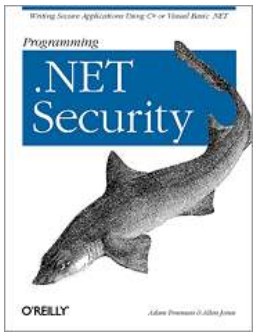
1. To learn more about the ProtectedConfigurationProvider Class, check out the MSDN Help topic, ProtectedConfigurationProvider Class.

2. To understand how Windows Data Protection works, check out: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/windataprotection-dpapi.asp

3. For an introduction to the cryptography classes in .NET, check out my article at: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhcvs03/html/vs03l1.asp

Wei-Meng Lee (weimenglee.blogspot.com) is a technologist and founder of Developer Learning Solutions, a technology company specializing in hands-on training of the latest Microsoft technologies.

Return to ONDotnet.com.



Sample Chapter

Read Online--Safari
Search this book on Safari:
Go
Only This Book
Code Fragments only

Search Windows

Go

Tagged Articles

Post to del.icio.us

This article has been tagged:
security connection database de

Articles that share the tag *security*:

Secure RSS Syndication (169 tags)

Google Your Site For Security Vulnerabilities (74 tags)

Building a Desktop Firewall (64 tags)

The Next 50 Years of Computer Security: An Interview with Alan Cox (42 tags)

Protect Yourself from WiFi Snoops (40 tags)

View All

Sponsored Resources

- Inside Lightroom

Related to this Article

Introducing Silverlight 1.1
by Todd Anglin
October 2007
$9.99 USD

AutoIt v3: Your Quick Guide
by Andy Flesner
September 2007
$7.99 USD