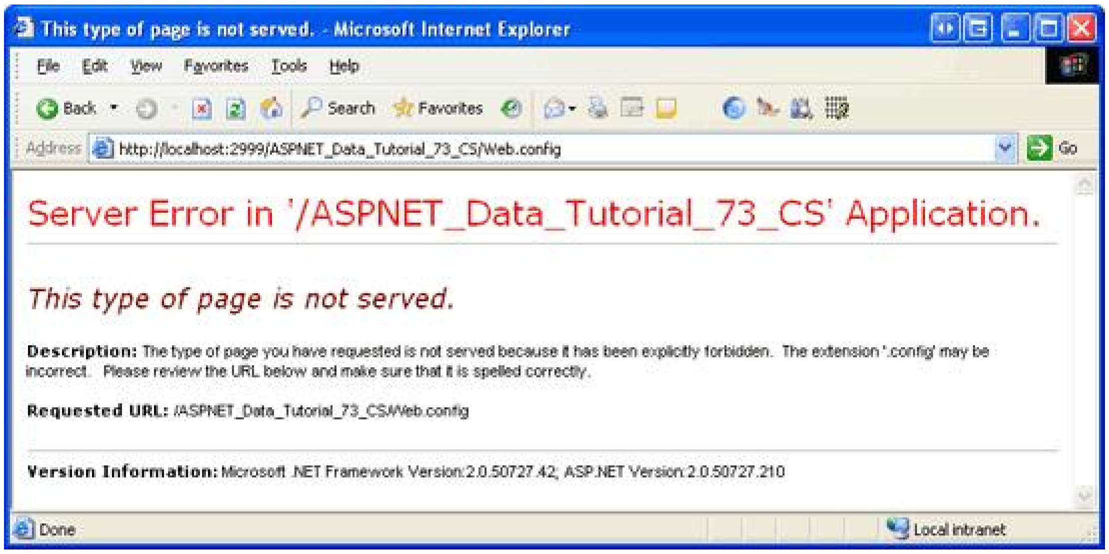
Working with Data in ASP.NET 2.0 :: Protecting Connection Strings and Other Configuration Information

Introduction

Configuration information for ASP.NET applications is commonly stored in an XML file named `Web.config`. Over the course of these tutorials we have updated the `Web.config` a handful of times. When creating the `Northwind` Typed DataSet in the first tutorial, for example, connection string information was automatically added to `Web.config` in the `<connectionStrings>` section. Later, in the Master Pages and Site Navigation tutorial, we manually updated `Web.config`, adding a `<pages>` element indicating that all of the ASP.NET pages in our project should use the `DataWebControls` Theme.

Since `Web.config` may contain sensitive data such as connection strings, it is important that the contents of `Web.config` be kept safe and hidden from unauthorized viewers. By default, any HTTP request to a file with the `.config` extension is handled by the ASP.NET engine, which returns the "*This type of page is not served*" message shown in Figure 1. This means that visitors cannot view your `Web.config` file's contents by simply entering 'http://www.YourServer.com/Web.config' into their browser's Address bar.


This type of page is not served. - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2999/ASPNET_Data_Tutorial_73_CS/Web.config Go
Done
Local intranet

Server Error in '/ASPNET_Data_Tutorial_73_CS' Application.

This type of page is not served.

Description: The type of page you have requested is not served because it has been explicitly forbidden. The extension '.config' may be incorrect. Please review the URL below and make sure that it is spelled correctly.

Requested URL: /ASPNET_Data_Tutorial_73_CS/Web.config

Version Information: Microsoft .NET Framework Version:2.0.50727.42; ASP.NET Version:2.0.50727.210

Figure 1: Visiting `Web.config` Through a Browser Returns a "This type of page is not served" Message

But what if an attacker is able to find some other exploit that allows her to view your `Web.config` file's contents? What could an attacker do with this information, and what steps can be taken to further protect the sensitive information within `Web.config`? Fortunately, most sections in `Web.config` do not contain sensitive information. What harm can an attacker perpetrate if they know the name of the default Theme used by your ASP.NET pages?

Certain `Web.config` sections, however, contain sensitive information that may include connection strings, user names, passwords, server names, encryption keys, and so forth. This information is typically found in the following `Web.config` sections:

- `<appSettings>`
- `<connectionStrings>`
- `<identity>`
- `<sessionState>`

In this tutorial we will look at techniques for protecting such sensitive configuration information. As we will see, the .NET Framework version 2.0 includes a protected configurations system that makes programmatically encrypting and decrypting selected configuration sections a breeze.

> **Note:** This tutorial concludes with a look at Microsoft's recommendations for connecting to a database from an ASP.NET application. In addition to encrypting your connection strings, you can help harden your system by ensuring that you are connecting to the database in a secure fashion.

Step 1: Exploring ASP.NET 2.0's Protected Configuration Options

ASP.NET 2.0 includes a protected configuration system for encrypting and decrypting configuration information. This includes methods in the .NET Framework that can be used to programmatically encrypt or decrypt configuration information. The protected configuration system uses the provider model, which allows developers to choose what cryptographic implementation is used.

The .NET Framework ships with two protected configuration providers:

- `RSAProtectedConfigurationProvider` - uses the asymmetric RSA algorithm for encryption and decryption.
- `DPAPIProtectedConfigurationProvider` - uses the Windows Data Protection API (DPAPI) for encryption and decryption.

Since the protected configuration system implements the provider design pattern, it is possible to create your own protected configuration provider and plug it into your application. See Implementing a Protected Configuration Provider for more information on this process.

The RSA and DPAPI providers use keys for their encryption and decryption routines, and these keys can be stored at the machine- or user-level. Machine-level keys are ideal for scenarios where the web application runs on its own dedicated server or if there are multiple applications on a server that need to share encrypted information. User-level keys are a more secure option in shared hosting environments where other applications on the same server should not be able to decrypt your application's protected configuration sections.

In this tutorial our examples will use the DPAPI provider and machine-level keys. Specifically, we will look at encrypting the `<connectionStrings>` section in `Web.config`, although the protected configuration system can be used to encrypt most any `Web.config` section. For information on using user-level keys or using the RSA provider, consult the resources in the Further Readings section at the end of this tutorial.

> **Note:** The `RSAProtectedConfigurationProvider` and `DPAPIProtectedConfigurationProvider` providers are registered in the `machine.config` file with the provider names `RsaProtectedConfigurationProvider` and `DataProtectionConfigurationProvider`, respectively. When encrypting or decrypting configuration information we will need to supply the appropriate provider name (`RsaProtectedConfigurationProvider` or `DataProtectionConfigurationProvider`) rather than the actual type name (`RSAProtectedConfigurationProvider` and `DPAPIProtectedConfigurationProvider`). You can find the `machine.config` file in the `$WINDOWS$\Microsoft.NET\Framework\`*`version`*`\CONFIG` folder.

Step 2: Programmatically Encrypting and Decrypting Configuration Sections

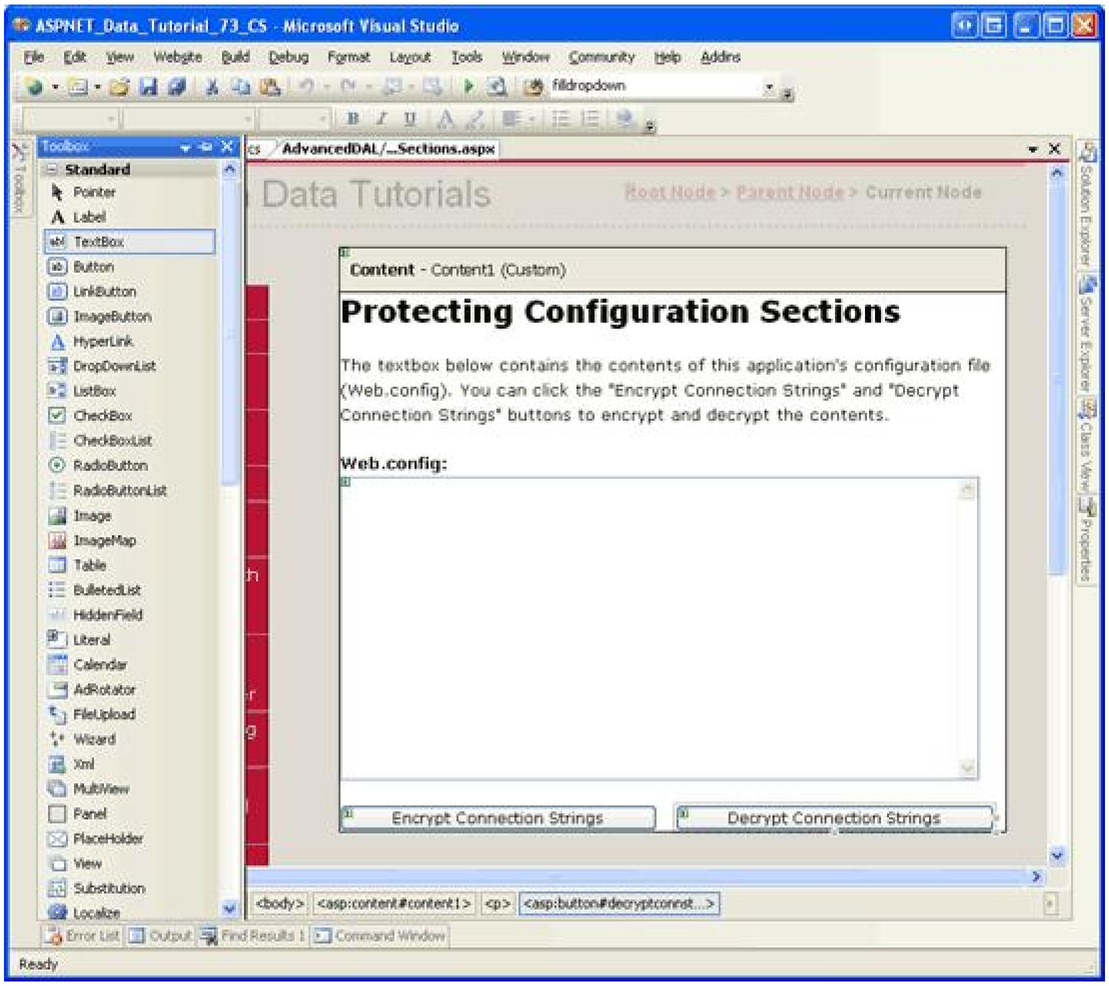
With a few lines of code we can encrypt or decrypt a particular configuration section using a specified provider. The code, as we will see shortly, simply needs to programmatically reference the appropriate configuration section, call its `ProtectSection` or `UnprotectSection` method, and then call the `Save` method to persist the changes. Moreover, the .NET Framework includes a helpful command line utility that can encrypt and decrypt configuration information. We will explore this command line utility in Step 3.

To illustrate programmatically protecting configuration information, let’s create an ASP.NET page that includes buttons for encrypting and decrypting the `<connectionStrings>` section in `Web.config`.

Start by opening the `EncryptingConfigSections.aspx` page in the `AdvancedDAL` folder. Drag a TextBox control from the Toolbox onto the Designer, setting its `ID` property to `WebConfigContents`, its `TextMode` property to `MultiLine`, and its `Width` and `Rows` properties to 95% and 15, respectively. This TextBox control will display the contents of `Web.config` allowing us to quickly see if the contents are encrypted or not. Of course, in a real application you would never want to display the contents of `Web.config`.

Beneath the TextBox, add two Button controls named `EncryptConnStrings` and `DecryptConnStrings`. Set their Text properties to “Encrypt Connection Strings” and “Decrypt Connection Strings”.

At this point your screen should look similar to Figure 2.


ASPNET_Data_Tutorial_73_CS - Microsoft Visual Studio
File Edit View Website Build Debug Format Layout Tools Window Community Help Addins
filldropdown
Toolbox
Standard
Pointer
Label
TextBox
Button
LinkButton
ImageButton
HyperLink
DropDownList
ListBox
CheckBox
CheckBoxList
RadioButton
RadioButtonList
Image
ImageMap
Table
BulletedList
HiddenField
Literal
Calendar
AdRotator
FileUpload
Wizard
Xml
MultiView
Panel
PlaceHolder
View
Substitution
Localize
AdvancedDAL/...Sections.aspx
Data Tutorials
Root Node > Parent Node > Current Node
Content - Content1 (Custom)
Encrypt Connection Strings
Decrypt Connection Strings
<body> <asp:content#content1> <p> <asp:button#decryptconnst...>
Error List Output Find Results 1 Command Window
Ready
Solution Explorer Server Explorer Class View Properties

Protecting Configuration Sections

The textbox below contains the contents of this application's configuration file (Web.config). You can click the "Encrypt Connection Strings" and "Decrypt Connection Strings" buttons to encrypt and decrypt the contents.

Web.config:

Figure 2: Add a TextBox and Two Button Web Controls to the Page

Next, we need to write code that loads and displays the contents of `Web.config` in the `WebConfigContents` TextBox when the page is first loaded. Add the following code to the page's code-behind class. This code adds a method named `DisplayWebConfig` and calls it from the `Page_Load` event handler when `Page.IsPostBack` is `false`:

```
protected void Page_Load(object sender, EventArgs e)
{
    // On the first page visit, call DisplayWebConfig method
    if (!Page.IsPostBack)
        DisplayWebConfig();
}

private void DisplayWebConfig()
{
    // Reads in the contents of Web.config and displays them in the TextBox
    StreamReader webConfigStream =
        File.OpenText(Path.Combine(Request.PhysicalApplicationPath, "Web.config"));
    string configContents = webConfigStream.ReadToEnd();
    webConfigStream.Close();

    WebConfigContents.Text = configContents;
}
```

The `DisplayWebConfig` method uses the `File` class to open the application's `Web.config` file, the `StreamReader` class to read its contents into a string, and the `Path` class to generate the physical path to the `Web.config` file. These three classes are all found in the `System.IO` namespace. Consequently, you will need to add a `using System.IO` statement to the top of the code-behind class or, alternatively, prefix these class names with "`System.IO.`".

Next, we need to add event handlers for the two Button controls' `Click` events and add the necessary code to encrypt and decrypt the `<connectionStrings>` section using a machine-level key with the DPAPI provider. From the Designer, double-click each of the Buttons to add a `Click` event handler in the code-behind class and then add the following code:

```
protected void EncryptConnStrings_Click(object sender, EventArgs e)
{
    // Get configuration information about Web.config
    Configuration config =
        WebConfigurationManager.OpenWebConfiguration(Request.ApplicationPath);

    // Let's work with the <connectionStrings> section
    ConfigurationSection connectionStrings = config.GetSection("connectionStrings");
    if (connectionStrings != null)
        // Only encrypt the section if it is not already protected
        if (!connectionStrings.SectionInformation.IsProtected)
        {
            // Encrypt the <connectionStrings> section using the
            // DataProtectionConfigurationProvider provider
            connectionStrings.SectionInformation.ProtectSection(
                "DataProtectionConfigurationProvider");
            config.Save();
```

```
                // Refresh the Web.config display
                DisplayWebConfig();
            }
    }

    protected void DecryptConnStrings_Click(object sender, EventArgs e)
    {
        // Get configuration information about Web.config
        Configuration config =
            WebConfigurationManager.OpenWebConfiguration(Request.ApplicationPath);

        // Let's work with the <connectionStrings> section
        ConfigurationSection connectionStrings =
            config.GetSection("connectionStrings");
        if (connectionStrings != null)
            // Only decrypt the section if it is protected
            if (connectionStrings.SectionInformation.IsProtected)
            {
                // Decrypt the <connectionStrings> section
                connectionStrings.SectionInformation.UnprotectSection();
                config.Save();

                // Refresh the Web.config display
                DisplayWebConfig();
            }
    }
```

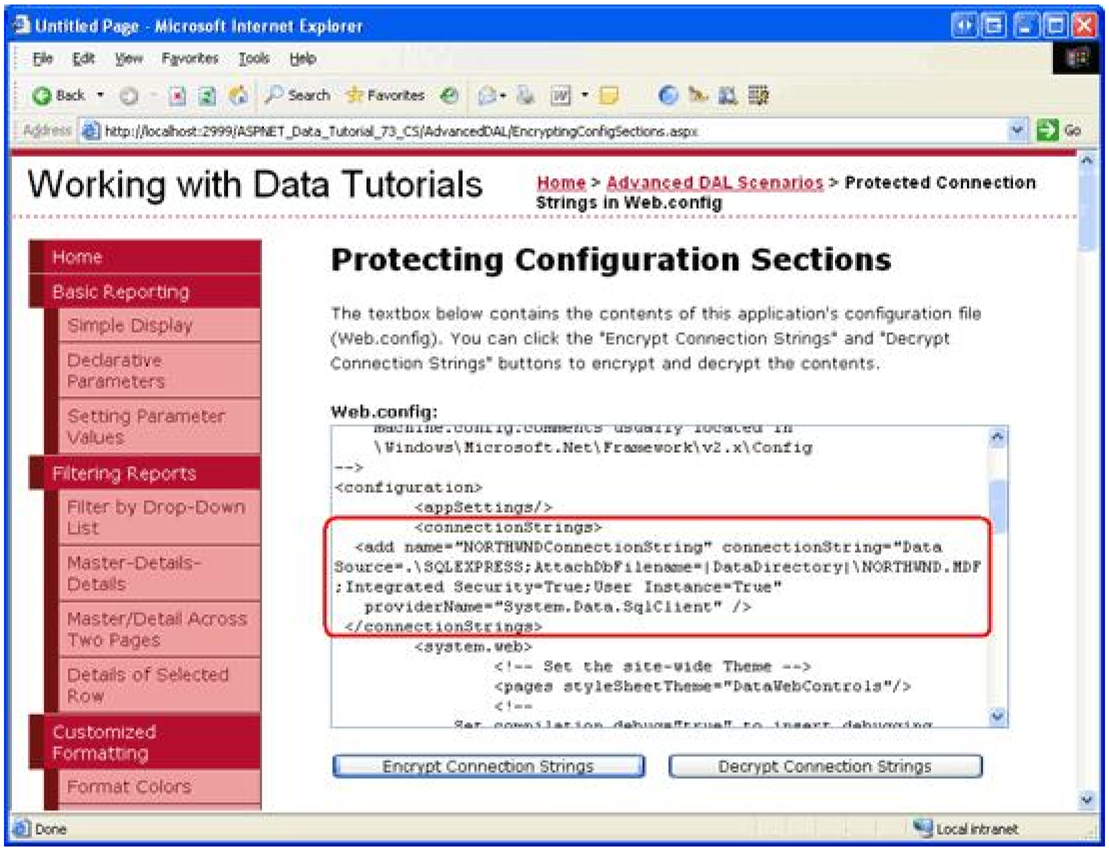
The code used in the two event handlers is nearly identical. They both start by getting information about the current application's `Web.config` file via the `WebConfigurationManager` class's `OpenWebConfiguration` method. This method returns the web configuration file for the specified virtual path. Next, the `Web.config` file's `<connectionStrings>` section is accessed via the `Configuration` class's `GetSection(`*`sectionName`*`)` method, which returns a `ConfigurationSection` object.

The `ConfigurationSection` object includes a `SectionInformation` property that provides additional information and functionality regarding the configuration section. As the code above shows, we can determine whether the configuration section is encrypted by checking the `SectionInformation` property's `IsProtected` property. Moreover, the section can be encrypted or decrypted via the `SectionInformation` property's `ProtectSection(`*`provider`*`)` and `UnprotectSection` methods.

The `ProtectSection(`*`provider`*`)` method accepts as input a string specifying the name of the protected configuration provider to use when encrypting. In the `EncryptConnString` Button's event handler we pass "DataProtectionConfigurationProvider" into the `ProtectSection(`*`provider`*`)` method so that the DPAPI provider is used. The `UnprotectSection` method can determine the provider that was used to encrypt the configuration section and therefore does not require any input parameters.

After calling the `ProtectSection(`*`provider`*`)` or `UnprotectSection` method, you must call the `Configuration` object's `Save` method to persist the changes. Once the configuration information has been encrypted or decrypted and the changes saved, we call `DisplayWebConfig` to load the updated `Web.config` contents into the TextBox control.

Once you have entered the above code, test it by visiting the `EncryptingConfigSections.aspx` page through a browser. You should initially see a page that lists the contents of `Web.config` with the `<connectionStrings>` section displayed in plain-text (see Figure 3).


Untitled Page - Microsoft Internet Explorer

http://localhost:2999/ASPNET_Data_Tutorial_73_CS/AdvancedDAL/EncryptingConfigSections.aspx

Working with Data Tutorials

Home > Advanced DAL Scenarios > Protected Connection Strings in Web.config

Protecting Configuration Sections

The textbox below contains the contents of this application's configuration file (Web.config). You can click the "Encrypt Connection Strings" and "Decrypt Connection Strings" buttons to encrypt and decrypt the contents.

Web.config:

```
    machine.config.comments usually located in
    \Windows\Microsoft.Net\Framework\v2.x\Config
-->
<configuration>
        <appSettings/>
        <connectionStrings>
  <add name="NORTHWNDConnectionString" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\NORTHWND.MDF
;Integrated Security=True;User Instance=True"
   providerName="System.Data.SqlClient" />
 </connectionStrings>
        <system.web>
                <!-- Set the site-wide Theme -->
                <pages styleSheetTheme="DataWebControls"/>
                <!--
            Set compilation debug="true" to insert debugging
```

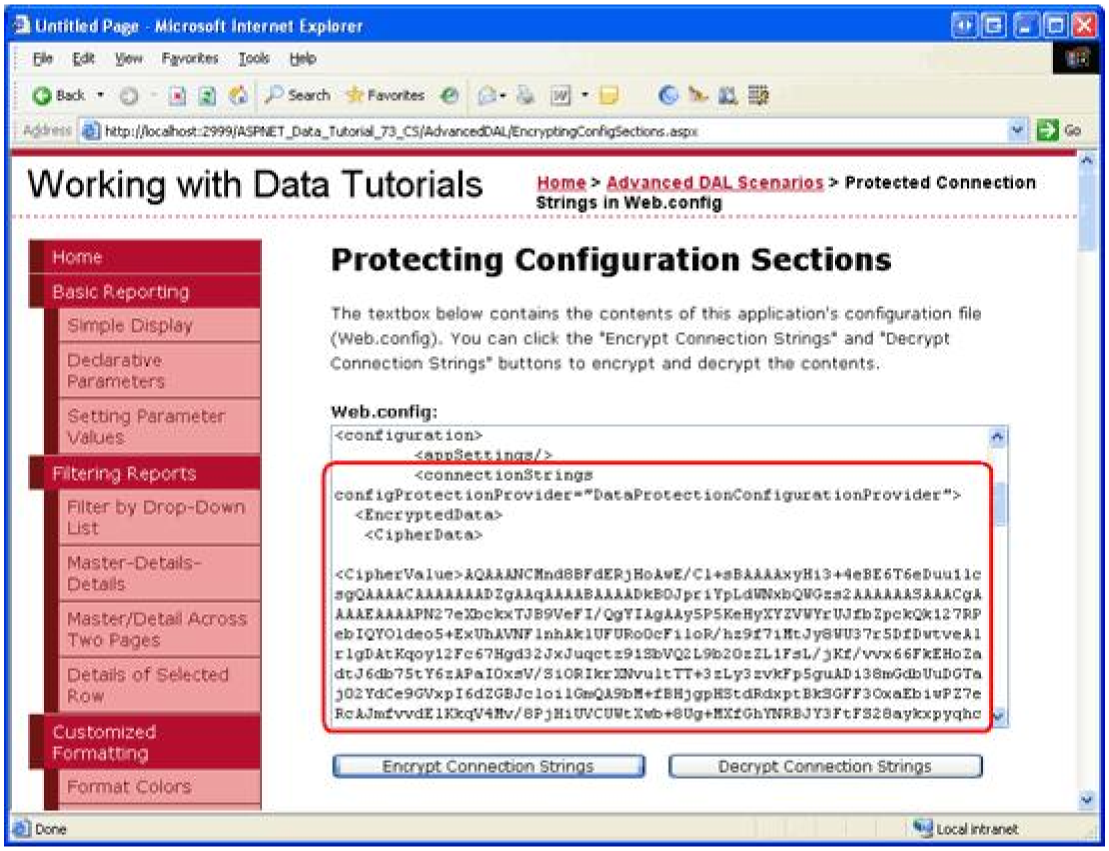
Figure 3: Add a TextBox and Two Button Web Controls to the Page

Now click the “Encrypt Connection Strings” button. If request validation is enabled, the markup posted back from the `WebConfigContents` TextBox will produce an `HttpRequestValidationException`, which displays the message, “A potentially dangerous `Request.Form` value was detected from the client.” Request validation, which is enabled by default in ASP.NET 2.0, prohibits postbacks that include un-encoded HTML and is designed to help prevent script-injection attacks. This check can be disabled at the page- or application-level. To turn it off for this page, set the `ValidateRequest` setting to `false` in the `@Page` directive. The `@Page` directive is found at the top of the page’s declarative markup.

```
<%@ Page ValidateRequest="False" ... %>
```

For more information on request validation, its purpose, how to disable it at the page- and application-level, as well as how to HTML encode markup, see Request Validation - Preventing Script Attacks.

After disabling request validation for the page, try clicking the “Encrypt Connection Strings” button again. On postback, the configuration file will be accessed and its `<connectionStrings>` section encrypted using the DPAPI provider. The TextBox is then updated to display the new `Web.config` content. As Figure 4 shows, the `<connectionStrings>` information is now encrypted.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2999/ASPNET_Data_Tutorial_73_CS/AdvancedDAL/EncryptingConfigSections.aspx
Go
Working with Data Tutorials
Home > Advanced DAL Scenarios > Protected Connection Strings in Web.config
Home
Basic Reporting
Simple Display
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Customized Formatting
Format Colors
Encrypt Connection Strings
Decrypt Connection Strings
Done
Local intranet

Protecting Configuration Sections

The textbox below contains the contents of this application's configuration file (Web.config). You can click the "Encrypt Connection Strings" and "Decrypt Connection Strings" buttons to encrypt and decrypt the contents.

Web.config:

```
<configuration>
        <appSettings/>
        <connectionStrings
configProtectionProvider="DataProtectionConfigurationProvider">
  <EncryptedData>
   <CipherData>

<CipherValue>AQAAANCMnd8BFdERjHoAwE/Cl+sBAAAAxyHi3+4eBE6T6eDuu1lc
sgQAAAACAAAAAAADZgAAqAAAABAAAADkBDJpriYpLdWNxbQWGzs2AAAAAASAAACgA
AAAEAAAAPN27eXbckxTJB9VeFI/QgYIAgAAy5P5KeHyXYZVWYrUJfbZpckQk127RP
ebIQYOldeo5+ExUhAVNFlnhAklUFURoOcFiloR/hz9f71MtJy8WU37r5DfDwtveAl
rlgDAtKqoy12Fc67Hgd32JxJuqctz91SbVQ2L9b20zZL1FsL/jKf/vvx66FkEHoZa
dtJ6db75tY6zAPaIOxsV/S1ORIkrXNvultTT+3zLy3zvkFp5guAD138mGdbUuDGTa
jO2YdCe9GVxpI6dZGBJclo1lGmQA9bM+fBHjgpHStdRdxptBk3GFF3OxaEbiwPZ7e
RcAJmfvvdE1KkqV4Mv/8PjHiUVCUWtXwb+8Ug+MXfGhYNRBJY3FtFS28aykxpyqhc
```

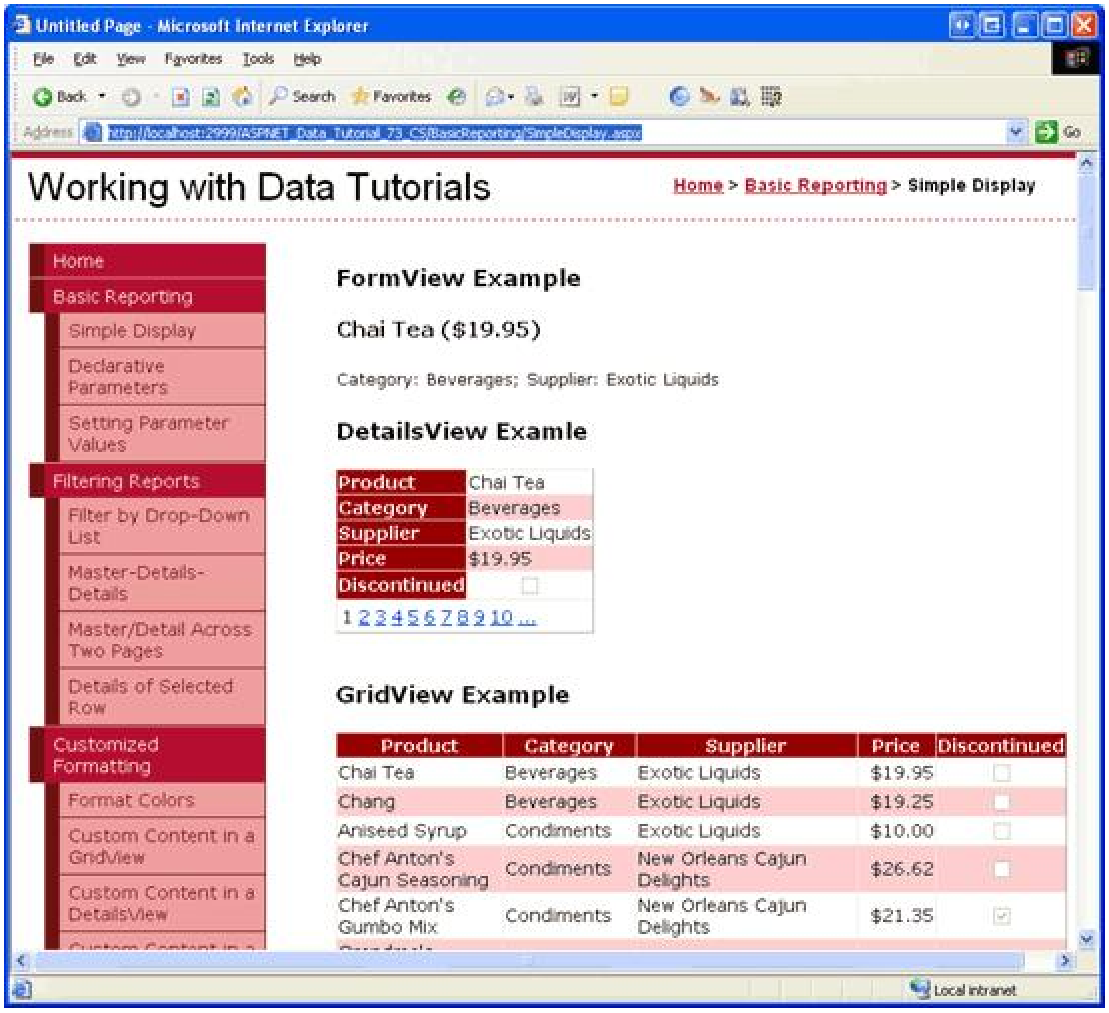
Figure 4: Clicking the "Encrypt Connection Strings" Button Encrypts the `<connectionString>` Section

The encrypted `<connectionStrings>` section generated on my computer follows, although some of the content in the `<CipherData>` element has been removed for brevity:

```
<connectionStrings
    configProtectionProvider="DataProtectionConfigurationProvider">
  <EncryptedData>
    <CipherData>
      <CipherValue>AQAAANCMnd8BFdERjHoAwE/...zChw==</CipherValue>
    </CipherData>
  </EncryptedData>
</connectionStrings>
```

Note: The `<connectionStrings>` element specifies the provider used to perform the encryption (`DataProtectionConfigurationProvider`). This information is used by the `UnprotectSection` method when the "Decrypt Connection Strings" button is clicked.

When the connection string information is accessed from `Web.config` - either by code we write, from a SqlDataSource control, or the auto-generated code from the TableAdapters in our Typed DataSets - it is automatically decrypted. In short, we do not need to add any extra code or logic to decrypt the encrypted `<connectionString>` section. To demonstrate this, visit one of the earlier tutorials at this time, such as the Simple Display tutorial from the Basic Reporting section (`~/BasicReporting/SimpleDisplay.aspx`). As Figure 5 shows, the tutorial works exactly as we would expect it, indicating that the encrypted connection string information is being automatically decrypted by the ASP.NET page.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:2999/ASPNET_Data_Tutorial_73_CS/BasicReporting/SimpleDisplay.aspx
Go
Working with Data Tutorials
Home > Basic Reporting > Simple Display
Home
Basic Reporting
Simple Display
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Customized Formatting
Format Colors
Custom Content in a GridView
Custom Content in a DetailsView
Local intranet

FormView Example

Chai Tea ($19.95)

Category: Beverages; Supplier: Exotic Liquids

DetailsView Examle

Product	Chai Tea
Category	Beverages
Supplier	Exotic Liquids
Price	$19.95
Discontinued	☐

1 2 3 4 5 6 7 8 9 10 ...

GridView Example

Product	Category	Supplier	Price	Discontinued
Chai Tea	Beverages	Exotic Liquids	$19.95	☐
Chang	Beverages	Exotic Liquids	$19.25	☐
Aniseed Syrup	Condiments	Exotic Liquids	$10.00	☐
Chef Anton's Cajun Seasoning	Condiments	New Orleans Cajun Delights	$26.62	☐
Chef Anton's Gumbo Mix	Condiments	New Orleans Cajun Delights	$21.35	✓

Figure 5: The Data Access Layer Automatically Decrypts the Connection String Information

To revert the `<connectionStrings>` section back to its plain-text representation, click the "Decrypt Connection Strings" button. On postback you should see the connection strings in `Web.config` in plain-text. At this point your screen should look like it did when first visiting this page (see in Figure 3).

Step 3: Encrypting Configuration Sections Using `aspnet_regiis.exe`

The .NET Framework includes a variety of command line tools in the `$WINDOWS$\Microsoft.NET\Framework\`*`version`*`\` folder. In the Using SQL Cache Dependencies tutorial, for instance, we looked at using the `aspnet_regsql.exe` command line tool to add the infrastructure necessary for SQL cache dependencies. Another useful command line tool in this folder is the ASP.NET IIS Registration tool (`aspnet_regiis.exe`). As its name implies, the ASP.NET IIS Registration tool is primarily used to register an ASP.NET 2.0 application with Microsoft's professional-grade Web server, IIS. In addition to its IIS-related features, the ASP.NET IIS Registration tool can also be used to encrypt or decrypt specified configuration sections in `Web.config`.

The following statement shows the general syntax used to encrypt a configuration section with the `aspnet_regiis.exe` command line tool:

```
aspnet_regiis.exe -pef section physical_directory -prov provider
```

section is the configuration section to encrypt (like “connectionStrings”), the *physical_directory* is the full, physical path to the web application’s root directory, and *provider* is the name of the protected configuration provider to use (such as “DataProtectionConfigurationProvider”). Alternatively, if the web application is registered in IIS you can enter the virtual path instead of the physical path using the following syntax:

```
aspnet_regiis.exe -pe section -app virtual_directory -prov provider
```

The following `aspnet_regiis.exe` example encrypts the `<connectionStrings>` section using the DPAPI provider with a machine-level key:

```
aspnet_regiis.exe -pef
"connectionStrings" "C:\Websites\ASPNET_Data_Tutorial_73_CS"
-prov "DataProtectionConfigurationProvider"
```

Similarly, the `aspnet_regiis.exe` command line tool can be used to decrypt configuration sections. Instead of using the `-pef` switch, use `-pdf` (or instead of `-pe`, use `-pd`). Also, note that the provider name is not necessary when decrypting.

```
aspnet_regiis.exe -pdf section physical_directory
  -- or --
aspnet_regiis.exe -pd section -app virtual_directory
```

Note: Since we are using the DPAPI provider, which uses keys specific to the computer, you must run `aspnet_regiis.exe` from the same machine from which the web pages are being served. For example, if you run this command line program from your local development machine and then upload the encrypted Web.config file to the production server, the production server will not be able to decrypt the connection string information since it was encrypted using keys specific to your development machine. The RSA provider does not have this limitation as it is possible to export the RSA keys to another machine.

Understanding Database Authentication Options

Before any application can issue `SELECT`, `INSERT`, `UPDATE`, or `DELETE` queries to a Microsoft SQL Server database, the database first must identify the requestor. This process is known as *authentication* and SQL Server provides two methods of authentication:

- **Windows Authentication** - the process under which the application is running is used to communicate with the database. When running an ASP.NET application through Visual Studio 2005’s ASP.NET Development Server, the ASP.NET application assumes the identity of the currently logged on user. For ASP.NET applications on Microsoft Internet Information Server (IIS), ASP.NET applications usually assume the identity of *`domainName`*`\`*`MachineName`* or *`domainName`*`\NETWORK SERVICE`, although this can be customized.
- **SQL Authentication** - a user ID and password values are supplied as credentials for authentication. With SQL authentication, the user ID and password are provided in the connection string.

Windows authentication is preferred over SQL authentication because it is more secure. With Windows authentication the connection string is free from a username and password and if the web server and database server reside on two different machines, the credentials are not sent over the network in plain-text. With SQL authentication, however, the authentication credentials are hard-coded in the connection string and are transmitted from the web server to the database server in plain-text.

These tutorials have used Windows authentication. You can tell what authentication mode is being used by inspecting the connection string. The connection string in `Web.config` for our tutorials has been:

```
Data Source=.\SQLEXPRESS; AttachDbFilename=|DataDirectory|\NORTHWND.MDF; Integrated
```

```
Security=True; User Instance=True
```

The “Integrated Security=True” and lack of a username and password indicate that Windows authentication is being used. In some connection strings the term “Trusted Connection=Yes” or “Integrated Security=SSPI” is used instead of “Integrated Security=True”, but all three indicate the use of Windows authentication.

The following example shows a connection string that uses SQL authentication. Note the credentials embedded within the connection string:

```
Server=serverName; Database=Northwind; uid=userID; pwd=password
```

Imagine that an attacker is able to view your application’s `Web.config` file. If you use SQL authentication to connect to a database that is accessible over the Internet, the attacker can use this connection string to connect to your database through SQL Management Studio or from ASP.NET pages on their own website. To help mitigate this threat, encrypt the connection string information in `Web.config` using the protected configuration system.

> **Note:** For more information on the different types of authentication available in SQL Server, see Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication. For further connection string examples illustrating the differences between Windows and SQL authentication syntax, refer to ConnectionStrings.com.

Summary

By default, files with a `.config` extension in an ASP.NET application cannot be accessed through a browser. These types of files are not returned because they may contain sensitive information, such as database connection strings, usernames and passwords, and so on. The protected configuration system in .NET 2.0 helps further protect sensitive information by allowing specified configuration sections to be encrypted. There are two built-in protected configuration providers: one that uses the RSA algorithm and one that uses the Windows Data Protection API (DPAPI).

In this tutorial we looked at how to encrypt and decrypt configuration settings using the DPAPI provider. This can be accomplished both programmatically, as we saw in Step 2, as well as through the `aspnet_regiis.exe` command line tool, which was covered in Step 3. For more information on using user-level keys or using the RSA provider instead, see the resources in the Further Reading section.

Happy Programming!

Further Reading

For more information on the topics discussed in this tutorial, refer to the following resources:

- Building Secure ASP.NET Application: Authentication, Authorization, and Secure Communication
- Encrypting Configuration Information in ASP.NET 2.0 Applications
- Encrypting `Web.config` Values in ASP.NET 2.0
- How To: Encrypt Configuration Sections in ASP.NET 2.0 Using DPAPI
- How To: Encrypt Configuration Sections in ASP.NET 2.0 Using RSA
- The Configuration API in .NET 2.0
- Windows Data Protection

About the Author

Scott Mitchell, author of seven ASP/ASP.NET books and founder of 4GuysFromRolla.com, has been working with Microsoft Web technologies since 1998. Scott works as an independent consultant, trainer, and writer. His latest book is *Sams Teach Yourself ASP.NET 2.0 in 24 Hours*. He can be reached at mitchell@4GuysFromRolla.com. or via his blog, which can be found at http://ScottOnWriting.NET.

Special Thanks To…

This tutorial series was reviewed by many helpful reviewers. Lead reviewers for this tutorial were Teresa Murphy and Randy Schmidt. Interested in reviewing my upcoming MSDN articles? If so, drop me a line at mitchell@4GuysFromRolla.com.