Using Forms Authentication in Reporting Services

Microsoft Corporation

March 2004

Applies To:
Microsoft® SQL Server™ 2000 Reporting Services

Summary: Learn about Reporting Services security extensions with a primary focus on Forms Authentication. In addition, download and deploy a sample Forms Authentication extension for Reporting Services. (23 printed pages)

To install the sample code, download the Forms Authentication Sample installer and run it on your computer.

Contents

Introduction

Deploying a secure, distributed enterprise reporting solution is a challenging process. From report access, to the data sources that supply important, sometimes sensitive data, you have some decisions to make regarding how to securely authenticate and authorize users in your reporting environment. And your reports are only as secure as the weakest link in the reporting chain.

The type of security you need depends on your reporting environment and the types of security systems already in place. Microsoft® Windows® Authentication is the primary system for securing reports in Microsoft® SQL Server™ 2000 Reporting Services. Windows Authentication offers tight integration with other Microsoft server products; because Reporting Services was designed and tested on Windows Authentication, it is most secure in this environment.

In certain cases, however, you may need to extend the Reporting Services security system to accommodate custom security in your enterprise. You can do this through the rich development platform of the Reporting Services API. This guide will present an overview of extensions in Reporting Services and security extensions in particular. You can also download and explore a sample Forms Authentication extension designed to work with Reporting Services. Afterwards, you should be able to take advantage of Reporting Services security extensions to add custom security to your enterprise reporting solution.

About This Guide

The information in this guide is designed to:

- Introduce you to Reporting Services security extensions.
- Identify where and how you need custom authentication and authorization in Reporting Services.
- Describe how authentication and authorization works in Reporting Services.
- Discuss Forms Authentication and how to implement it.
- Provide you with a Forms Authentication sample that you can download and explore.

What You Need to Know

This guide is not an introduction to ASP.NET security or Forms Authentication. It will not provide you with in-depth knowledge of programming or application security. As a developer looking to implement a security extension for Reporting Services, you should already have in-depth experience with one or more of the following:

- Microsoft Reporting Services features and capabilities, specifically authentication, authorization, and role-based security.
- The Microsoft .NET Framework.
- ASP.NET and ASP.NET security.
- Forms Authentication.
- Development experience in a .NET language. The sample is only available in C# at this time.

If you want to dive right into the code, you can skip ahead to the "Forms Authentication Sample" section. However, you might find the beginning sections helpful in introducing you to some of the technologies you will be working with and how they all fit together.

Reporting Services Platform

Reporting Services enables the design, deployment, and delivery of reports throughout the enterprise. From a developer's perspective, Reporting Services offers a wide variety of programming opportunities through the key developer platforms of the .NET Framework and Web services. Reporting Services can be deployed "out of the box" to provide a comprehensive reporting solution in most any company. However, the open and extensible programming architecture of Reporting Services make it less of an off-the-shelf product and more of a reporting platform for developers and end users alike.

Extending Reporting Services

Reporting Services was designed to be extensible. Managed code APIs enable you to develop, install, and manage extensions consumed by many Reporting Services components. You can create private or shared assemblies using the .NET Framework and add new Reporting Services functionality to meet evolving business needs. Developers can extend Reporting Services in the following ways:

- Create data processing extensions in addition to the Microsoft SQL Server, Oracle, and OLE DB providers that currently ship with Reporting Services. Data processing extensions can be designed to read data from your own unique data sources and can be used to incorporate additional business logic in the creation and filtering of sets of data.
- Create delivery extensions in addition to the E-mail and File Share delivery extensions that currently ship with Reporting Services. Delivery extensions can be used to deliver reports to devices including fax machines, pagers, printers, and more.
- Create rendering extensions in addition to those that already ship with Reporting Services.
- Create security extensions in addition to the Windows Authentication extension, which is currently the default security mechanism of the product.

As mentioned previously, this guide focuses on extending the security system of Reporting Services through Forms Authentication.

Security Extensions

A Reporting Services security extension enables the authentication and authorization of users or groups; that is, it enables different users to log into a report server and, based on their identities, perform different tasks or operations. By default, Reporting Services uses a Windows-based authentication extension, which uses Windows account protocols to verify the identities of users who claim to have accounts on the system. Reporting Services uses a role-based security system to authorize users. The Reporting Services role-based security model is similar to the role-based security models of other technologies. Because security extensions are based on an open and extensible API, you can create new authentication and authorization extensions in Reporting Services. The following is an example of a typical security extension implementation that uses Forms-based authentication and authorization:

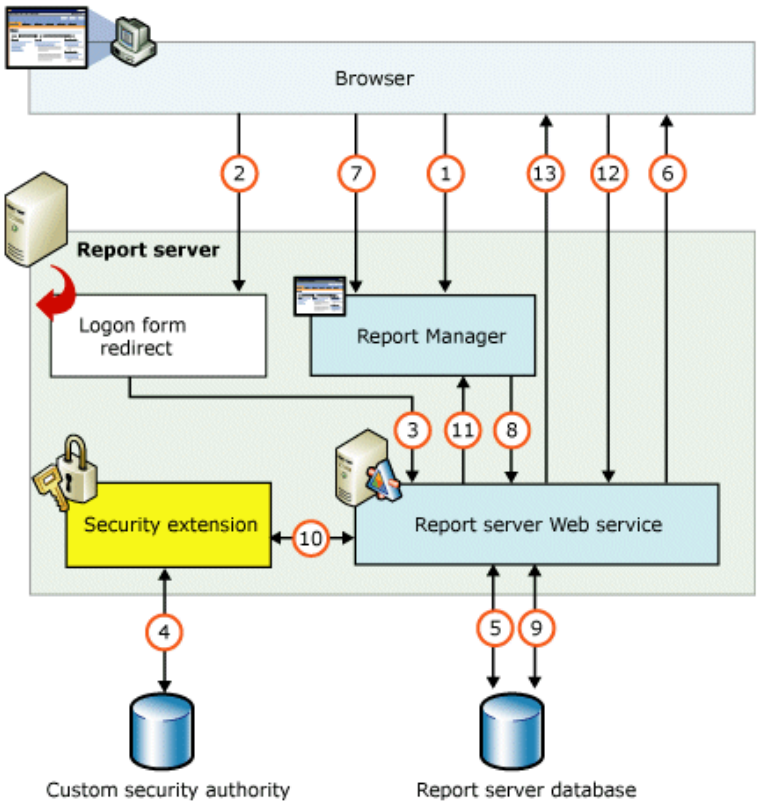


Figure 1

As shown in the illustration, authentication and authorization take place as follows:

1. A user attempts to access Report Manager by entering a URL and is redirected to a form that collects user credentials for the client application.
2. The user submits credentials to the form.

3. The user credentials are submitted to the Reporting Services Web service through the **LogonUser** method.
4. The Web service calls the customer-supplied security extension and verifies that the user name and password exist in the custom security authority.
5. Upon authentication, the Web service creates an authentication ticket (known as a "cookie"), manages the ticket, and verifies the user's role for the Home page of Report Manager.
6. The Web service returns the cookie to the browser and displays the appropriate user interface in Report Manager.
7. Once authenticated, the browser makes requests to Report Manager while transmitting the cookie in the HTTP header. These requests are in response to user actions within the Report Manager application.
8. The cookie is transmitted in the HTTP header to the Web service along with the requested user operation.
9. The cookie is validated, and if it is valid, the report server returns the security descriptor and other information relating to the requested operation from the report server database.
10. If the cookie is valid, the report server makes a call to the security extension to check if the user is authorized to perform the specific operation.
11. If the user is authorized, the report server performs the requested operation and returns control to the caller.
12. Once the user is authenticated, URL access to the report server utilizes the same cookie. The cookie is transmitted in the HTTP header.
13. The user continues to request operations on the report server until the session has ended.

When to Implement a Security Extension

Microsoft recommends that you use Windows Authentication if at all possible. However, custom authentication and authorization for Reporting Services may be appropriate in the following two cases:

- You have an internet or extranet application that does not and cannot use Windows accounts.
- You have custom-defined users and roles and need to provide a matching authorization scheme in Reporting Services.

Security Extension API

The following table describes the available interfaces and classes for security extensions.

Interface or class	Description
IAuthenticationExtension Interface	Represents an authentication extension in Reporting Services and enables you to implement a security extension class that can be used to authenticate users using a custom authentication scheme.
IAuthorizationExtension Interface	Represents an extension that can be used to extend the authorization feature of Reporting Services and enables you to implement a security extension class that can be used to authorize users to perform operations.
IExtension Interface	Represents an extension in Reporting Services.
AceCollection Class	Represents a collection of access control entries specifying access rights for one or more trustees.
AceStruct Class	An access control entry for a trustee (user, group, or computer) that specifies the operations that a trustee can perform on items in the report server database.
CatalogOperationsCollection Class	Represents a collection of catalog operations.
DatasourceOperationsCollection Class	Represents a collection of data source operations.
FolderOperationsCollection Class	Represents a collection of folder operations.
OperationNames Class	Contains the field names and corresponding values for operations that users can perform on items in Reporting Services.
ReportOperationsCollection Class	Represents a collection of report operations.
ResourceOperationsCollection Class	Represents a collection of resource operations.

For more information about the various interfaces and classes of the security extension API, see Reporting Services Books Online.

Authentication in Reporting Services

Authentication is the process of establishing a user's right to an identity. There are many techniques that you can use to authenticate a user. The most common way is to use passwords. When you implement Forms Authentication, for example, you want an implementation that queries users for credentials (usually by some interface that requests a login name and password) and then validates users against a user store, such as a database table or configuration file. If the credentials can't be validated, the authentication process fails and the user will assume an anonymous identity.

In Reporting Services, the Windows operating system handles the authentication of users either through integrated security or through the explicit reception and validation of user credentials. Custom authentication in Reporting Services can be developed to support additional authentication schemes. This is made possible through the security extension interface **IAuthenticationExtension**. All extensions inherit from **IExtension** the base interface for any extension deployed and used by the report server. **IExtension** as well as **IAuthenticationExtension** are members of the **Microsoft.ReportingServices.Interfaces** namespace.

The heart of any authentication in Reporting Services is the **LogonUser** method. This member of the Reporting Services Web service can be used to pass user credentials to a report server for validation. Your underlying security extension implements **IAuthenticationExtension.LogonUser** which contains your custom authentication code. In the Forms Authentication sample, discussed later

in this guide, **LogonUser** performs an authentication check against the supplied credentials and a custom user store in a database. In the Forms Authentication sample, it looks like this:

In AuthenticationExtension.cs (Forms Authentication Sample)

```
public bool LogonUser(string userName, string password, string authority)
{
   return AuthenticationUtilities.VerifyPassword(userName, password);
}
```

In AuthenticationUtilities.cs (Forms Authentication Sample)

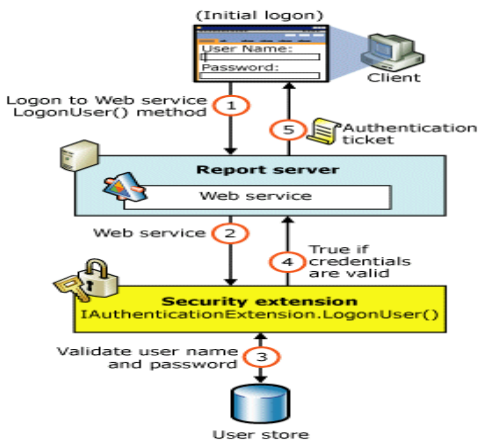
```
internal static bool VerifyPassword(string suppliedUserName,
   string suppliedPassword)
{
   bool passwordMatch = false;
   // Get the salt and pwd from the database based on the user name.
   // See "How To: Use DPAPI (Machine Store) from ASP.NET," "How To:
   // Use DPAPI (User Store) from Enterprise Services," and "How To:
   // Create a DPAPI Library" for more information about how to use
   // DPAPI to securely store connection strings.
   SqlConnection conn = new SqlConnection(
      "Server=localhost;" +
      "Integrated Security=SSPI;" +
      "database=UserAccounts");
   SqlCommand cmd = new SqlCommand("LookupUser", conn);
   cmd.CommandType = CommandType.StoredProcedure;

   SqlParameter sqlParam = cmd.Parameters.Add("@userName",
       SqlDbType.VarChar,
       255);
   sqlParam.Value = suppliedUserName;
   try
   {
      conn.Open();
      SqlDataReader reader = cmd.ExecuteReader();
      reader.Read(); // Advance to the one and only row
      // Return output parameters from returned data stream
      string dbPasswordHash = reader.GetString(0);
      string salt = reader.GetString(1);
      reader.Close();
      // Now take the salt and the password entered by the user
      // and concatenate them together.
      string passwordAndSalt = String.Concat(suppliedPassword, salt);
      // Now hash them
      string hashedPasswordAndSalt =
         FormsAuthentication.HashPasswordForStoringInConfigFile(
         passwordAndSalt,
         "SHA1");
      // Now verify them. Returns true if they are equal
      passwordMatch = hashedPasswordAndSalt.Equals(dbPasswordHash);
   }
   catch (Exception ex)
   {
       throw new Exception("Exception verifying password. " +
          ex.Message);
   }
   finally
   {
       conn.Close();
   }
   return passwordMatch;
}
```

Authentication Flow

The Reporting Services Web service provides custom authentication to enable Forms Authentication by Report Manager and the report server.

The **LogonUser** method of the Reporting Services Web service is used to submit credentials to the report server for authentication. The Web service uses HTTP headers to pass an authentication ticket (known as a "cookie") from the server to the client for validated logon requests.

The following illustration depicts the method of authenticating users to the Web service when your application is deployed with a report server configured to use a custom authentication extension.



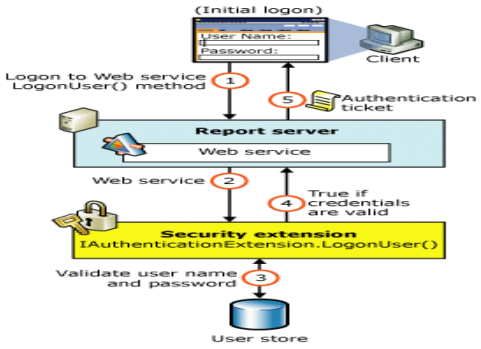


Figure 2

As shown in Figure 2, the authentication process is as follows:

1. A client application calls the Web service method **LogonUser** to authenticate a user.
2. The Web service makes a call to the **LogonUser** method of your security extension, specifically, the class that implements **IAuthenticationExtension**.
3. Your implementation of **LogonUser** validates the user name and password in the user store or security authority.
4. Upon successful authentication, the Web service creates a cookie and manages it for the session.
5. The Web service returns the authentication ticket to the calling application on the HTTP header.

When the Web service successfully authenticates a user through the security extension, it generates a cookie that is used for subsequent requests. The cookie may not persist within the custom security authority because the report server does not own the security authority. The cookie is returned from the **LogonUser** Web service method and is used in subsequent Web service method calls and in URL access.

> **Security** In order to avoid compromising the cookie during transmission, authentication cookies returned from **LogonUser** should be transmitted securely using Secure Sockets Layer (SSL) encryption.

If you access the report server through URL access when a custom security extension is installed, Internet Information Services (IIS) and ASP.NET automatically manage the transmission of the authentication ticket. If you are accessing the report server through the SOAP API, your implementation of the proxy class must include additional support for managing the authentication ticket. For more information about using the SOAP API and managing the authentication ticket, see "Using the Web Service with Custom Security" later in this guide.

Authorization in Reporting Services

Authorization is the process of determining whether an identity should be granted the requested type of access to a given resource in the report server database. Reporting Services uses a role-based authorization architecture that grants a user access to a given resource based on the user's role in the application. Security extensions for Reporting Services contain an implementation of an authorization component that is used to grant access to users once they are authenticated on the report server. Authorization is invoked when a user attempts to perform an operation on the system or a report server item through the SOAP API and through URL access. This is made possible through the security extension interface **IAuthorizationExtension**. As stated previously, all extensions inherit from **IExtension** the base interface for any extension that you deploy. **IExtension** and **IAuthorizationExtension** are members of the **Microsoft.ReportingServices.Interfaces** namespace.

In authorization, the key to any custom security implementation is the access check, which takes place in the method **CheckAccess**. **CheckAccess** is called each time a user attempts an operation on the report server. The **CheckAccess** method is overloaded for each operation type. For folder operations, an example of an access check might look like the following:

```
// Overload for Folder operations
public bool CheckAccess(
   string userName,
   IntPtr userToken,
   byte[] secDesc,
   FolderOperation requiredOperation)
{
   // If the user is the administrator, allow unrestricted access.
   if (userName == m_adminUserName)
      return true;

   AceCollection acl = DeserializeAcl(secDesc);
   foreach(AceStruct ace in acl)
   {
         if (userName == ace.PrincipalName)
         {
            foreach(FolderOperation aclOperation in
               ace.FolderOperations)
            {
               if (aclOperation == requiredOperation)
                     return true;
            }
         }
   }
   return false;
}
```

The report server calls the **CheckAccess** method by passing in the name of the logged-on user, a user token, the security descriptor for the item, and the requested operation. Here you would check the security descriptor for the user name and the appropriate permission to complete the request, then return **true** to signify that access is granted or **false** to signify access is denied.

Security Descriptors

When setting authorization policies on items in the report server database, a client application (such as Report Manager) submits the user information to the security extension along with a security policy for the item. This security policy and user information are known collectively as a security descriptor. A security descriptor contains the following information for an item in the report server database:

- The group or user that has some type of permission to perform operations on the item.
- The item's type.
- A discretionary access control list controlling access to the item.

Security descriptors are created using the Web service methods **SetPolicies** and **SetSystemPolicies**. For more information regarding these methods and the Web service, see Reporting Services Books Online.

Authorization Flow

Reporting Services authorization is controlled by the security extension currently configured to run on the server. Authorization is role-based and limited to the permissions and operations supplied by the Reporting Services security architecture. The following diagram depicts authorizing users to operate on items in the report server database:

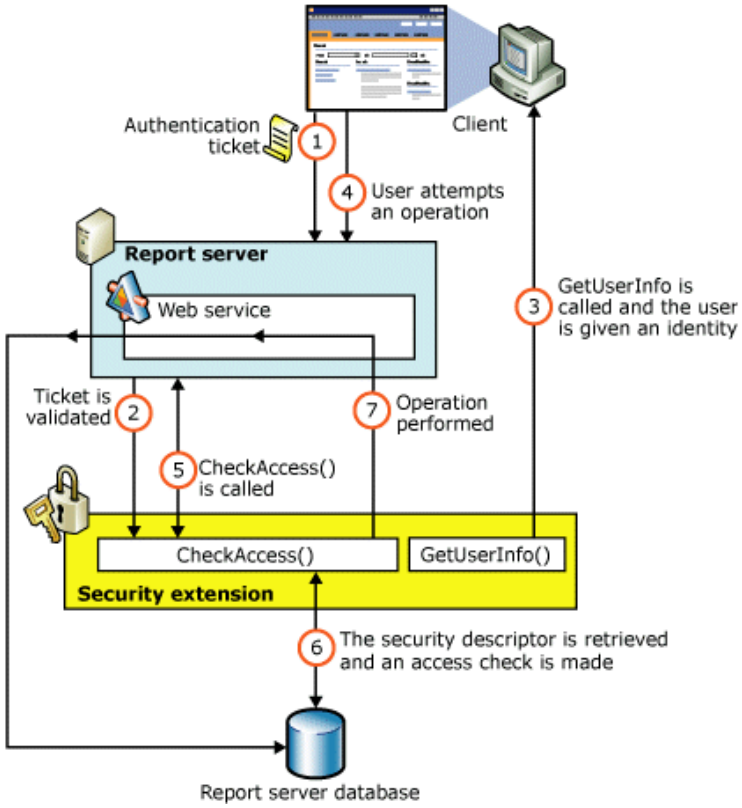


Figure 3

As shown in Figure 3, authorization follows this sequence:

1. Once authenticated, client applications make requests to the report server through the Reporting Services Web service methods. An authentication ticket is passed to the report server in the form of a cookie in the HTTP header of each Web request.
2. The cookie is validated prior to any access check.
3. Once the cookie is validated, the report server calls **GetUserInfo** and the user is given an identity.
4. The user attempts an operation through the Reporting Services Web service.
5. The report server calls the **CheckAccess** method.
6. The security descriptor is retrieved and passed to a custom security extension implementation of **CheckAccess**. At this point the user, group, or computer is compared to the security descriptor of the item being accessed and is authorized to perform the requested operation.
7. If the user is authorized, the Web service performs the operation and returns a response to the calling application.

Forms Authentication Sample

If you haven't already, you should download the Forms Authentication Sample installer and run it on your computer to install the sample code.

About the Sample

The Forms Authentication sample security extension, which is available for download as part of this guide, uses Forms Authentication along with SQL Server to provide a custom security model that works with Reporting Services. This sample is provided for educational purposes only. It is not intended to be used in a production environment and has not been tested in a production environment. Microsoft does not provide technical support for this sample. In some cases, best practices (such as the creation of one-way hash, salted passwords) are demonstrated in the sample. In other cases, best practice has been side-stepped in favor of simplicity. It is important to note that much of the setup and management of the sample security extension require administrator access on the report server computer. In any event, using this sample on a production server in a connected environment is not recommended.

If you have any questions or concerns regarding Reporting Services security extensions, please contact Microsoft Consulting Services (MCS), Premier Support Services (PSS), or another Microsoft support services representative.

Requirements

You must have the following to use this sample:

- A report server with Microsoft SQL Server 2000 Reporting Services installed.
- Microsoft Internet Explorer 6.0 or later for accessing Report Manager.
- A development computer with Microsoft® Visual Studio® .NET 2003.

Considerations

Consider the following when implementing a custom security extension or when using the sample:

- It is not possible to run a report server under a mixed-mode security system (for example, both Windows and Forms Authentication). This is true of any ASP.NET application.
- Remember to save backup copies of all configuration files that you change as a result of configuring this sample.
- Although possible, reverting to Windows Authentication after deploying the sample can be difficult. See "Removing the Sample Extension" later in this guide for more information.
- Overriding Windows Authentication is risky. The Report Server was designed to be extensible with regards to security, but is fully tested and supported using Windows Authentication only.
- Always use Secure Sockets Layer (SSL) with Forms Authentication.
- In this sample, user input is passed to Transact-SQL commands for authentication. You should take great care, in your own custom security extensions that use Forms Authentication with SQL Server, to validate user input and to ensure that the resulting commands do not contain syntax errors. You want to make sure to validate all user input so that a malicious user can't cause your application to run arbitrary SQL commands (also known as a "SQL injection attack"). Validating the supplied user name during a logon process is particularly important because your report server custom security model depends entirely on being able to correctly identify, authenticate, and authorize users.
- In your custom user store, avoid allowing users to enter names with the following characters: : ? ; @ & = + $, \ * > < | . " /. User names with these characters may cause problems with the My Reports feature, because folders are created in the server using the name of the user, and these characters can cause problems with folder names and folder paths. The sample code uses regular expressions to test for valid user names and to ensure that the path name will not exceed the maximum allowed path length. You should perform similar validation in your custom authentication code.

Forms Authentication

Forms Authentication is a type of ASP.NET authentication in which an unauthenticated user is directed to an HTML form. Once the user provides credentials, the system issues a cookie containing an authentication ticket. On later requests, the system first checks the cookie to see if the user was already authenticated by the report server.

Reporting Services does not inherently support the use of Forms Authentication. However, Reporting Services can be extended to support Forms Authentication using the security extensibility interfaces available through the Reporting Services API. If you extend Reporting Services to use Forms, use Secure Sockets Layer (SSL) for all communications with the report server to prevent malicious users from gaining access to another user's cookie. SSL enables clients and a report server to authenticate each other and to ensure that no other computers can read the contents of communications between the two computers. All data sent from a client through an SSL connection is encrypted so that malicious users cannot intercept passwords or data sent to a report server.

Forms Authentication is generally implemented to support non-Windows accounts and authentication. A graphical interface is presented to a user who requests access to a report server, and the supplied credentials are submitted to a security authority for authentication.

Forms Authentication schemes work when an interactive user is present to enter credentials. However, for unattended applications that communicate directly with the Reporting Services Web service, Forms Authentication must be combined with a custom authentication scheme.

Forms Authentication is appropriate for Reporting Services when:

- You need to store and authenticate users that do not have Microsoft Windows accounts, and
- You need to provide your own user interface form as a logon page between different pages on a Web site.

Consider the following when writing a custom security extension that supports Forms Authentication:

- If you use Forms Authentication, anonymous access must be enabled on the report server virtual directory in Internet Information Services (IIS).
- ASP.NET authentication must be set to Forms. You configure ASP.NET authentication in the Web.config file for the report server.
- Reporting Services can authenticate and authorize users with either Windows authentication or custom authentication. Reporting Services does not support simultaneous use of multiple security extensions.

Deploying the Sample

Before you can run and examine the code for the Forms Authentication sample, several steps must be taken. After you follow the steps for installation and configuration, you can use or debug the sample on your report server, or you can simply view the sample code in Visual Studio .NET.

Compiling and Installing the Extension Assembly

You must follow these steps to compile and install the extension. The steps assume that you have installed Reporting Services to the default location: C:\Program Files\Microsoft SQL Server\MSSQL\Reporting Services. This location will be referred to throughout the remainder of this guide as <*install*>.

To compile the sample using Visual Studio .NET

1. Open CustomSecurity.sln in Microsoft Visual Studio .NET 2003. If you downloaded the source code and installed the sample to the default location, you can access it at C:\Program Files\Microsoft SQL Server\MSSQL\Reporting Services\Samples\Extensions.
2. In **Solution Explorer**, select the **CustomSecurity** project.
3. On the **Project** menu, click **Add Reference**.
4. The **Add References** dialog box opens.
5. Click the **.NET** tab.
6. Click **Browse**, and navigate to find Microsoft.ReportingServices.Interfaces on your local drive. By default, the assembly is located in the <*install*>\ReportServer\bin directory. Click **OK**.

   The selected reference is added to your project.
7. On the **Build** menu, click **Build Solution**.
8. Copy Microsoft.Samples.ReportingServices.CustomSecurity.dll and Microsoft.Samples.ReportingServices.CustomSecurity.pdb to the <*install*>\ReportServer\bin directory.
9. Copy Microsoft.Samples.ReportingServices.CustomSecurity.dll and Microsoft.Samples.ReportingServices.CustomSecurity.pdb to the <*install*>\ReportManager\bin directory.
10. Copy the Logon.aspx page to the <*install*>\ReportServer directory and copy the UILogon.aspx page to the <*install*>\ReportManager\Pages directory.

Adding the Extension to the Configuration Files

After the assembly and logon pages are copied to the server, you need to make some modifications to the Report Server and Report Manager configuration files.

Important Make backup copies of all of your configuration files before making any changes.

To modify the RSReportServer.config file

1. Open the RSReportServer.config file with Visual Studio .NET or a simple text editor such as Notepad. RSReportServer.config is located in the <*install*>\ReportServer directory.
2. Locate the <**Security**> and <**Authentication**> elements and modify the settings as follows:

```
<Security>
   <Extension Name="Forms"
Type="Microsoft.Samples.ReportingServices.CustomSecurity.Authorization,
Microsoft.Samples.ReportingServices.CustomSecurity" >
      <Configuration>
         <AdminConfiguration>
            <UserName>username</UserName>
         </AdminConfiguration>
      </Configuration>
   </Extension>
</Security>
<Authentication>
   <Extension Name="Forms"
Type="Microsoft.Samples.ReportingServices.CustomSecurity.AuthenticationExtension,
 Microsoft.Samples.ReportingServices.CustomSecurity" />
</Authentication>
```

To modify the RSWebApplication.config file

1. Next, you need to open the Report Manager configuration file, RSWebApplication.config, located in the <*install*>\ReportManager directory.
2. Locate the <**UI**> element and update it as follows:

```
<UI>
   <CustomAuthenticationUI>
      <loginUrl>/Pages/UILogon.aspx</loginUrl>
         <UseSSL>True</UseSSL>
   </CustomAuthenticationUI>
   <ReportServerUrl>http://<server>/ReportServer</ReportServerUrl>
</UI>
```

> **Security** If you are running the sample security extension in a development environment that does not have a SSL certificate installed, you must change the value of the **<UseSSL>** element to False in the previous configuration entry. Microsoft recommends that you always use SSL when combining Reporting Services with Forms Authentication.

Adding Security Policies for the Extension

You will need to add a code group for your custom security extension that grants **FullTrust** permission for your extension. You do this by adding the code group to the rssrvpolicy.config file located the <*install*>\ReportServer directory. Locate the existing code group in the security policy file that has a URL membership of $CodeGen as indicated below and then add an entry as follows to the rssrvpolicy.config:

```
<CodeGroup
   class="UnionCodeGroup"
   version="1"
   PermissionSetName="FullTrust">
   <IMembershipCondition
      class="UrlMembershipCondition"
      version="1"
      Url="$CodeGen$/*"
   />
</CodeGroup>
<CodeGroup
   class="UnionCodeGroup"
   version="1"
   Name="SecurityExtensionCodeGroup"
   Description="Code group for the sample security extension"
   PermissionSetName="FullTrust">
   <IMembershipCondition
      class="UrlMembershipCondition"
      version="1"
      Url="C:\Program Files\Microsoft SQL Server\MSSQL\Reporting Services\ReportServer\bin\Microsoft.Samples.ReportingServices.CustomSecurity.dll"
   />
</CodeGroup>
```

> **Note** For simplicity, the Forms Authentication Sample is weak-named and requires a simple URL membership entry in the security policy files. In your production security extension implementation, you should create strong-named assemblies and use the strong name membership condition when adding security policies for your assembly. For more information about strong-named assemblies, see "Creating and Using Strong-Named Assemblies" on MSDN at http://msdn.microsoft.com/library/en-us/cpguide/html/cpconworkingwithstrongly-namedassemblies.asp.

Next, you will need to increase the permissions for the MyComputer code group in the Report Manager policy file, rsmgrpolicy.config, located in the <*install*>\ReportManager directory. Locate the following code group in rsmgrpolicy.config and change the **PermissionSetName** attribute from **Execution** to **FullTrust** as follows:

```
<CodeGroup
        class="FirstMatchCodeGroup"
        version="1"
        PermissionSetName="FullTrust"
        Description="This code group grants MyComputer code Execution
permission. ">
    <IMembershipCondition
            class="ZoneMembershipCondition"
            version="1"
            Zone="MyComputer" />
```

Configuring the Web.config Files

To use Forms Authentication, you need to modify the Web.config files for Report Manager and Report Server to change the authentication mode and disable impersonation.

To modify the Web.config file for Report Server

1. Open the Web.config file in a text editor. By default, the file is located in the <*install*>\ReportServer directory.
2. Locate the <**identity**> element and set the **impersonate** attribute to **false**.

   ```
   <identity impersonate="false" />
   ```

3. Locate the <**authentication**> element and change the **mode** attribute to **Forms**.
4. Add the following <**forms**> element as a child of the <**authentication**> element and set the **loginUrl**, **name**, **timeout**, and **path** attributes as follows:

   ```
   <authentication mode="Forms">
      <forms loginUrl="logon.aspx" name="sqlAuthCookie" timeout="60"
                  path="/"></forms>
      </authentication>
   ```

5. Add the following <**authorization**> element directly after the <**authentication**> element.

```
<authorization>
   <deny users="?" />
</authorization>
```

This will deny unauthenticated users the right to access the report server. The previously established **loginUrl** attribute of the <**authentication**> element will redirect unauthenticated requests to the Logon.aspx page.

To modify the Web.config file for Report Manager

1. Open the Web.config for Report Manager. It is located in the <*install*>\ReportManager directory.
2. Disable impersonation by locating the section `<identity impersonate= "true" />` and changing it to the following: `<identity impersonate="false" />`.

Configuring Anonymous Authentication

By default, the Windows user group **Guests** includes the IUSR_*computername* account. This account is used to initially log on locally and view the Logon.aspx page. To support Forms Authentication, you must enable anonymous access for the ReportServer virtual directory. By default, anonymous access is disabled.

To enable anonymous authentication

1. In **Internet Information Services**, select the ReportServer virtual directory, usually a member of the Default Web site, and open its property tabs.
2. Click the **Directory Security** tab.
3. In the **Anonymous access and authentication control** section, click **Edit**.

   The **Authentication Methods** dialog box appears.
4. Select the **Anonymous access** check box.
5. Click **OK**.
6. Repeat the above steps for the Reports virtual directory.

Creating the User Account Database

The sample includes a database script, createuserstore.sql, that enables you to set up a user store for the Forms sample in a SQL Server database.

To create the UserAccounts database

1. Open **Query Analyzer**, and then connect to your local instance of SQL Server.
2. Locate the createuserstore.sql SQL script file. The script file is contained within the sample project files. Note that you must replace "LocalMachine" with your own computer name towards the end of the script. For Windows 2003 users, replace LocalMachine\ASPNET with NT AUTHORITY\NETWORK SERVICE (except when in IIS 5 compatibility mode)
3. Run the query to create the **UserAccounts** database.
4. Exit Query Analyzer.

Testing the Sample

The following procedure tests the sample extension. You will register an administrator user, which adds the user name, password hash, and salt value to the users table in the **UserAccounts** database. It also will require you to enter that user name in the Report Server Configuration File. You will then log the same user on to ensure the correct operation of the password verification routines as well as the proper loading of the extension assembly by the report server.

To test the sample security extension deployment

1. Restart IIS by running iisreset.exe at the command prompt.
2. Open Report Manager. You can do this from the Reporting Services program menu or by accessing the Reports virtual directory from your browser.
3. Enter a user name and password and click **Register User** to add the user to the accounts database.
4. Open the RSReportServer.config file. Locate the <**Security**> element and add the previously registered user name as follows:

```
<Security>
   <Extension Name="Forms"
Type="Microsoft.Samples.ReportingServices.CustomSecurity.Authorization,
Microsoft.Samples.ReportingServices.CustomSecurity" >
      <Configuration>
         <AdminConfiguration>
            <UserName>username</UserName>
         </AdminConfiguration>
      </Configuration>
   </Extension>
</Security>
```

5. Return to the UILogon.aspx page, re-enter the user name and password, and then click **Logon**.

You should have access to Report Manager and the report server with no restrictions. The administrator user that you create has equivalent

permissions on the report server to those of a Built-in administrator account on the local computer. For the purpose of this sample, you can only have one user designated as an administrator. Once you have a built-in administrator account, you can register additional users and assign them roles in the report server.

> **Note** It is recommended that you add your administrator user to the official System Administrator and Content Manager (root folder) roles of your report server. This prevents empty security descriptors from existing in the report server database. For more information about the System Administrator and Content Manager roles, see Reporting Services Books Online.

Using the Web Service with Custom Security

You can use the Web service API with Forms Authentication just as you would with Windows Authentication. However, you must call **LogonUser** in your Web service code and pass the credentials of the current user. In addition, your Web service client will not have the benefit of automatic cookie management, which is provided by Internet Explorer or other Web browsers. You will have to extend the **ReportingService** proxy class to include cookie management. This can be done by overriding the **GetWebRequest** and **GetWebResponse** methods of the Web service class.

For an example of this, see "ReportingService.LogonUser Method" on MSDN at http://msdn.microsoft.com/library/en-us/rsprog/htm/rsp_ref_soapapi_service_lz_3d7q.asp.

Debugging the Sample Extension

Running the sample extension in the debugger is not only a great way to troubleshoot difficulties you may have, but it is also an effective way to step through the code and see the report server authentication and authorization process as it is happening.

The Microsoft .NET Framework provides several debugging tools that can help you analyze the sample code. The tool that works best will depend on what you are trying to accomplish. For the purpose of this guide, the debugging tool of choice is Visual Studio .NET 2003.

To debug the Forms Authentication sample code

1. Be sure to follow the previous steps and deploy the sample.
2. Launch Visual Studio .NET 2003 and open CustomSecurity.sln on your test report server.
3. Open Internet Explorer and navigate to Report Manager while leaving the sample code open in Visual Studio.
4. Navigate to Visual Studio and the custom security extension project, and set some break points in the code.
5. With the extension project still the active window, click **Process** on the **Debug** menu.

   The **Processes** dialog opens.
6. From the list of processes, select the aspnet_wp.exe process (or w3wp.exe if your application is deployed on IIS 6.0), and click **Attach**. When the **Attach to Process** dialog opens, make sure that the program type **Common Language Runtime** is selected, and then click **OK**. For improved debugging performance, make sure that **Native** is not a selected program type.
7. Now, enter user credentials in the logon form and click "Logon.". If code that corresponds to your break points is hit, the debugger should stop execution at your first break point.
8. Step through your code using the F11 key. For more information about using Visual Studio for debugging, see your Visual Studio .NET documentation.

   > **Note** Debugging this way requires a lot of resources and processor time. If you run into difficulties, close Visual Studio, reset IIS, and begin again by attaching the CustomSecurity solution to the ASP.NET worker process and logging on to Report Manager.

Removing the Sample Extension

While not generally recommended, it is possible to revert back to Windows Authentication after you have tried out the sample. To revert to Windows security, do the following:

- Restore the following files from your backup copies: Web.config, RSReportServer.config, and RSWebApplication.config. This should set the authentication and authorization methods for the report server to the default Windows security. This should also remove any entries you made for your extension in the Report Server or Report Manager configuration files.
- Disable anonymous access in Internet Information Services (IIS) for the report server virtual directory.

After the configuration information is removed, your security extension is no longer available to the report server. You should not have to remove any security descriptors that were created while you were running the report server under the sample security extension. The report server automatically assigns the **System Administrator** role to the BUILTIN\Administrators group on the computer hosting the report server when Windows Authentication is enabled. However, you will have to manually re-apply any role-based security for your Windows users.

Again, reverting back to Windows Authentication after migrating to a different security extension is not generally recommended. If you do, you may experience errors when you attempt to access items in the report server database that have custom security descriptors, but no Windows Authentication security descriptors.

Conclusion

Microsoft SQL Server 2000 Reporting Services is an extensible reporting platform for the development and deployment of enterprise reports. Reporting Services includes a Windows Authentication module that uses Windows accounts to secure access to the report server, but in some cases, you may need to support a different security module. To accomplish this, Reporting Services includes a set of security extensibility APIs

that support the creation of a custom security extension. In the sample provided, you saw one approach to implementing custom security through Forms Authentication. Microsoft is working to meet evolving security scenarios in the enterprise by making security extensibility available in this and future releases of Reporting Services. As a developer, you should be aware of the risks of implementing a custom security extension and should carefully consider the exact implementation and application of the security system used by your report server.

Additional Resources

For more information, see the following resources on MSDN:

- "How To: Use Forms Authentication with SQL Server 2000"
- "Forms Authentication Provider"
- "Using Role-Based Security"
- "Microsoft.ReportingServices.Interfaces Namespace"