Architectural Design for Security

ROLE Base Security Management

By: Rajendra Man Banepali
BCA, MSCIT
URL: www.rmb.com.np
Email: info@rmb.com.np

Version: 1.0.0

Date: 18-July-2010 [First Edition]

Download Source Code: www.rmb.com.np

1 Contents

2 Foreword

In computer systems security, role-based access control (RBAC) is an approach to restricting system access to authorized users. It is a newer alternative approach to mandatory access control (MAC) and discretionary access control (DAC). RBAC is sometimes referred to as role-based security.

3 RBAC Model

Within an organization, roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. Members of staff (or other system users) are assigned particular roles, and through those role assignments acquire the permissions to perform particular system functions. Since users are not assigned permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of simply assigning appropriate roles to the user; this simplifies common operations, such as adding a user, or changing a user's department.

Three primary rules are defined for RBAC:

1. Role assignment: A subject can execute a transaction only if the subject has selected or been assigned a role.

2. Role authorization: A subject's active role must be authorized for the subject. With rule 1 above, this

3. Transaction authorization: A subject can execute a transaction only if the transaction is authorized for the subject's active role. With rules 1 and 2, this rule ensures that users can execute only transactions for which they are authorized.

4 Getting start

ASP.NET schema is used for this application. Let's prepare ASP.NET schema.

4.1 Adding the SqlMembershipProvider Schema to the Database

The **SqlMembershipProvider** requires a particular set of tables, views, and stored procedures to be installed in the user store database. These requisite database objects can be added using the aspnet_regsql.exe tool. This file is located in the %WINDIR%\Microsoft.Net\Framework\v2.0.50727\ folder.

Note: The aspnet_regsql.exe tool offers both command line functionality and a graphical user interface. The graphical interface is more users friendly and is what we will examine in this tutorial. The command line interface is useful when the addition of the **SqlMembershipProvider** schema needs to be automated, such as in build scripts or automated testing scenarios.

The aspnet_regsql.exe tool is used to add or remove ASP.NET application services to a specified SQL Server database. The ASP.NET application services encompass the schemas for the **SqlMembershipProvider** and SqlRoleProvider, along with the schemas for the SQL-based providers for other ASP.NET 2.0 frameworks. We need to provide two bits of information to the aspnet_regsql.exe tool:

- Whether we want to add or remove application services, and
- The database from which to add or remove the application services schema

In prompting for the database to use, the aspnet_regsql.exe tool asks us to provide the name of the server the database resides on, the security credentials for connecting to the database, and the database name. If you are using the non-Express Edition of SQL Server, you should already know this information, as it is the same information you must provide through a connection string when working with the database through an ASP.NET web page. Determining the server and database name when using a SQL Server 2005 Express Edition database in the **App_Data** folder, however, is a bit more involved.

4.2 Problem

You are building an Intranet expense report application for your organization, and want to enable role-based authentication and authorization capabilities within it. Specifically, you want to create logical roles called “approvers”, “auditors”, and “administrators” for the application, and grant/deny end-users access to functionality within the application based on whether they are in these roles.

[**If you are trying to use Active Directory for user authentication and role, you have to use “Windows” Authentication Mode in web.config. Now we are trying with FORMS Authentication. No code will be change in FORMS and Windows Authentication so you can use both authentications.**] - Because your application is an Intranet solution, you want to use Windows Authentication to login the users accessing the application (avoiding them having to manually login). However, because the roles you want to define are specific to your application, you do not want to define or store them within your network’s Windows Active Directory. Instead, you want to define and store these roles within a database. You then want to map Windows user accounts stored within Active Directory to these roles, and grant/deny access within the application based on them.

In addition to using roles to authorize access to individual pages within the application, you want to dynamically filter the links displayed within the site’s menu navigation based on whether users have permissions (or not) to those links. And lastly, you want to build-in a custom role-management administration UI directly within the expense report application for “expense app administrators” to manage these roles and control who has access to the capabilities of the app:

4.3 Discussion

ASP.NET provides a flexible authentication management system that allows you to easily take advantage of **Forms** authentication to identify “who” the authenticated user accessing your site is. You can learn how to enable this and how it works by reviewing my previous Recipe: Enabling **Forms** Authentication within an Intranet ASP.NET Web Application.

Most web applications typically have at least dozens (if not thousands or millions) of authenticated users accessing them. Authorizing access to pages or functionality within a site is difficult (if not impossible) when managing it on an individual user account level. Instead, it is much better for the application developer to define higher level “roles” to map users into, and then grant/deny access or permissions based on these roles.

For example: if we were building an internal expense reporting application, we might want to create three roles for the application: “approvers”, “auditors” and “administrators”. We would then only allow users in the “approvers” role to access the portion of the site that enables someone to approve

employee expenses. We would only allow users in the "auditors" role to access the portion of the site that generates reports and analysis on employee spending. And we would only allow the select few users in the "administrators" role to have access to the admin pages of the application and administrator can also create user.

By coding against roles like "approvers", "auditors" and "administrators", as opposed to individual account names, we can have very clean code within our application, and make the application very flexible as we add/remove users to the system and change their permissions over time. For example, if "Saral" gets promoted to be a manager with expense approver permissions, all that needs to happen for him to get access to the approver portion of the expense app is for the administrator to go in and add him into the "approvers" role – no code changes or configuration changes to the app need to be made.

[**If you are trying to use Active Directory for user authentication and role, you have to use "Windows" Authentication Mode in web.config. Now we are trying with FORMS Authentication. No code will be change in FORMS and Windows Authentication so you can use both authentications.**] - ASP.NET supports multiple places where user to role mappings can be stored and defined. When Windows Authentication is enabled, ASP.NET will by default use the Active Directory user/group mappings to support role access permission checks. This is useful when the permission checks you want to perform are global to your company environment.

For many applications, though, you might want to implement more local role policies – where the roles you define are specific to the application. In cases like these you either might not want to store these in a central Active Directory store, or your network administrator might not even allow it. Instead, you might want to store the role definitions and user mappings locally within a database – while still using Windows Authentication to identify and login the users stored within them.

The below walkthrough demonstrates how to-do this, and builds a complete end-to-end application to illustrate how all the pieces fit together.

4.4 Step1: Create a New Web Site

We'll then add a new file called “Web.SiteMap” in the top level "root" directory of the project. SiteMap files enable us to specify the site hierarchy and link structure that we want to use to organize pages within our site, and can be used to databind Menus and navigation UI against (treeviews, breadcrumb controls, etc). Within the Web.SiteMap file add this XML to define the link structure for the site we are going to build:

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >

  <siteMapNode url="default.aspx" title="Home">
    <siteMapNode url="approver.aspx" title="Approver Expenses"
roles="Approvers" />
    <siteMapNode url="Audit" title="Audit Page" roles="Auditors" />

    <siteMapNode url="Administrator\AddUser.aspx" title="Add User"
roles="Admins" />
    <siteMapNode url="Administrator\AlterRole.aspx" title="Alter Role"
roles="Admins" />
    </siteMapNode>

</siteMap>
```

Then we'll create a Master Page template called “MasterPage.master” at the root/Shared of the project. This will enable us to define an overall consistent layout to use for all pages on the site. Within it we'll add a Treeview control that is databound to our site navigation hierarchy that we defined above, and which will provide a hierachical menu for navigating the pages in our site. Define the contents of the Site.Master page like so:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="Shared_MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Expense Report</title>
    <link href="Template.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="form1" runat="server">

    <div class="header">

        <h1>
            Expense Report Sample
            <asp:LoginName ID="LoginName1" FormatString="(Welcome {0})"
CssClass="loginname" runat="server" />
        </h1>

    </div>

    <div class="menu">

        <asp:LoginStatus ID="LoginStatus1" runat="server" />

        <asp:TreeView ID="TreeView1" DataSourceID="SiteMapDataSource1"
            EnableViewState="False" runat="server" ImageSet="Arrows">
            <ParentNodeStyle Font-Bold="False" />
            <HoverNodeStyle Font-Underline="True" ForeColor="#5555DD" />
            <SelectedNodeStyle Font-Underline="True" ForeColor="#5555DD"
                HorizontalPadding="0px" VerticalPadding="0px" />
            <NodeStyle Font-Names="Tahoma" Font-Size="10pt" ForeColor="Black"
                HorizontalPadding="5px" NodeSpacing="0px"
VerticalPadding="0px" />
        </asp:TreeView>

        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />

    </div>

    <div class="content">

        <asp:contentplaceholder id="MainContent" runat="server">
        </asp:contentplaceholder>

    </div>
    </form>
</body>
</html>
```

Note: for convenience sake I'm using the "Simple" auto-format selection option within the VS designer to style the TreeView control above. One downside is that this is embedding inline styles. To enable a pure CSS styling solution for the TreeView above I would want to download and use the ASP.NET 2.0 CSS Control Adapter Toolkit.

After we define the Site.Master template, we'll create a new page called "Default.aspx" that is based on the master-page and that we will use as the home-page for the site. Create the page like so:

```
<%@ Page Title="" Language="C#" MasterPageFile="~/Shared/MasterPage.master"
AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" Runat="Server">
    <p>
    <a href="mailto:You@Home">You@Home</a><br />
    Rajendra Man Banepali</p>
</asp:Content>
```

Note that the tree-view's site navigation links on the left-hand side of the page are databound from the ASP.NET 2.0 Site Navigation System.

Next create two subdirectories in the project called "administrator" and "audit" and add pages within these sub-directories called "AlterUser.aspx, AddUser.aspx" and "Default.aspx" respectively. Also then create a top-level page called "approver.aspx" in the root directory of the site. All of these pages should be based on the MasterPage.master template we defined above.

4.5 Step 2: Enable FORMS Authentication

To enable FORMS Authentication for our expense report web-site above, and force users to always be authenticated when visiting the application, we'll want to open our web.config file at the root of the project and add this XML to it:

```
<authentication mode="FORMS" />
<authorization>
   <deny users="?"/>
</authorization>
```

Once we've configured the settings above, every user to the site will automatically be validated and authenticated via our FORMS authentication system on the network.

4.6 Step 3: Enable SQL based Role Management

ASP.NET 2.0 ships with built-in role manager providers that work against SQL Express, SQL Server, and Active Directory (which can also be used against ADAM stores). If you'd prefer to use your own custom database (or the file-system or your own LDAP system), you can also build your own role manager provider and easily add it to the system. This web-site (http://msdn.microsoft.com/asp.net/downloads/providers) details how the ASP.NET 2.0 Provider Model works, how you can build your own providers to plug-in, and enables you to download the source code for the built-in providers that ASP.NET ships with.

For the purposes of this sample we are going to use either SQL Express or SQL Server to store our role mappings. To begin with we'll want to enable the ASP.NET role-manager in our web.config file. We'll do this by adding this section within the

<system.web> group:

```
<roleManager enabled="true"/>
```

If you have SQL Express installed on your machine, then you are done. ASP.NET will automatically provision a new SQL Express database within your app_data folder at runtime that has the appropriate Roles tables configured to persist the role mappings we'll do. You don't need to take any additional steps to configure this.

If you don't have SQL Express installed, and instead want to use a SQL Server to store the Roles data, you'll need to create a database within SQL to store the ASP.NET Application Service tables, and update your web.config file to point at the database. Below is a sample configuration entry that shows how to configure the web.config file to use a SQL database:

```
<roleManager enabled="true" defaultProvider="SqlRoleManager">

  <providers>
    <clear/>
    <add name="SqlRoleManager"
         type="System.Web.Security.SqlRoleProvider"
         connectionStringName="SqlRoleManagerConnection"
         applicationName="sahara" />
  </providers>

</roleManager>
```

Where the "SqlRoleManagerConnection" connection-string we referenced above is defined within the <connectionStrings> section of our web.config file like so:

```
<connectionStrings>
  <add name="SqlRoleManagerConnection"
       connectionString="Data Source=localhost;Initial
Catalog=BasePageSample;Integrated Security=SSPI;">
  </add>
</connectionStrings>
```

Important: If you explicitly declare a new provider reference like I did within the <providers> section above, you need to make sure that you specify the “applicationName” attribute.

And now, when we run our application again, ASP.NET will automatically authenticate the incoming users to the site using FORMS Authentication, and use the SQL Database we defined above to retrieve all role mappings.

4.7 Step 4: Create Our Roles and Initial User Mappings to Them

Now that we have configured our role provider, we can use the "Roles" API (the System.Web.Security.Roles class) within ASP.NET to create roles, and manage users within them.

One tip/trick I like to use is to take advantage of the "Application_Start" event handler within Global.asax to setup my Roles if they don't already exist, and map any initial users into them if necessary. To-do this, choose File->Add New Item and select the "Global.asax" file item. Then add this code to your Application_Start event handler:

```
void Application_Start(object sender, EventArgs e)
    {
        // Code that runs on application startup

        if (Roles.RoleExists("Auditors") == false)
            Roles.CreateRole("Auditors");

        if (Roles.RoleExists("Approvers") == false)
            Roles.CreateRole("Approvers");

        if (Roles.RoleExists("Admins") == false)
            Roles.CreateRole("Admins");

        MembershipCreateStatus status;

        MembershipUser newUser = Membership.CreateUser(
            "rajendra",
            "sahara",
            "eccplsu7@hotmail.com",
            "Who is who", "Sahara is my Inspiration",
            true,
            out status);

        //Membership.DeleteUser("rajendra", true);
        if (!Roles.IsUserInRole("rajendra", "Admins"))
        Roles.AddUserToRole("rajendra", "Admins");

        //*** Adding user as user
        newUser = Membership.CreateUser(
            "user",
            "user",
            "rajendra.banepali@live.com",
            "Who is who", "Sahara is my Inspiration",
            true,
            out status);

        //Membership.DeleteUser("rajendra", true);
```

```
        if (!Roles.IsUserInRole("user", "Approvers"))
            Roles.AddUserToRole("user", "Approvers");
    }//**
```

In addition there is Membership API used for user membership handling.

This event handler gets called once every-time the ASP.NET application starts up. As you can see above, within it I am checking to see whether our three roles exist in the configured Roles database – and if not I create them. This is a good approach you can use to setup the initial admin users for your application.

Now, when we run the application again, it will automatically provision the three new roles into the database and add me into the admin one (note: in the code above I've only added myself to the admin role – I'm not yet in the Auditors or Approvers role).

4.8 Step 5: Authorizing Access Based On Roles

To verify that our roles were setup correctly, let's add some authorization rules to grant/deny access to portions of the site based on them. I can do this using the "authorization" section of our web.config files.

First create new "web.config" files within both the "Administraor" and "Audit" directories:

Then add the below XML content within the web.config file within the "Administror" directory:

```
<?xml version="1.0"?>
<configuration>
   <system.web>
       <authorization>
          <allow roles="Admins"/>
          <deny users="*"/>
       </authorization>
   </system.web>
</configuration>
```

This tells ASP.NET to allow users within the "Admins" role to access the pages within the directory, but to block everyone else who isn't in this role. We'll then want to add similar content to the web.config file within the "audit" directory (which does the same logical thing as above – except in this case only allowing "Auditors" access).

For the "Approver.aspx" page that is in the root directory of the application we'll want to-do something a little extra. Because it isn't in a sub-directory of its own, we can't use a global directory rule like we did above. Instead, we'll want to restrict the rule to only apply to the "Approver.aspx" page within that directory. We can do this by specifying a <location path="url"> directive around it within our root web.config file on the site:

```
<location path="Approver.aspx">
  <system.web>
    <authorization>
      <allow roles="Approvers"/>
      <deny users="*"/>
    </authorization>
  </system.web>

</location>
```

And now when we run the application again, and try to access the "Audit Page" or "Approver Page" within the site we'll get this error message.

This happens because when we created the roles in our Application_Start event handler we didn't add ourselves into the "Approvers" or "Auditors" role – and so ASP.NET is denying us access to those resources (like it should). When we click the "Admin" link we are able to access it, though, because we belong to the "Admins" role.

4.9 Step 6: Implementing Security Trimming for our Menu

One issue you will have noticed when we ran the sample above is that the menu on the left-hand side of the screen is still displaying all of the links when we visit the site – including links to the "Audit" and "Approver" pages that we don't currently have access to (because we aren't in those roles).

Ideally we want to hide those links and only display them to users within the appropriate roles to access them. This avoids users inadvertently seeing our "Access Denied" error message above. The good news is that this is easy to implement using a cool ASP.NET 2.0 feature called "Security Trimming".

To implement security trimming, add this XML section to your web.config file:

```
<siteMap defaultProvider="XmlSiteMapProvider" enabled="true">
  <providers>
    <add name="XmlSiteMapProvider"
      description="Default SiteMap provider."
      type="System.Web.XmlSiteMapProvider "
      siteMapFile="Web.sitemap"
      securityTrimmingEnabled="true" />
  </providers>
</siteMap>
```

This tells ASP.NET to enable security-trimming at the Site Navigation provider. Once we do this, we can then open up our [web.sitemap] configuration file again and update the individual nodes within it with "roles" attributes indicating what nodes are visible to incoming users:

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >

  <siteMapNode url="default.aspx" title="Home">
    <siteMapNode url="approver.aspx" title="Approver Expenses"
roles="Approvers" />
    <siteMapNode url="Audit" title="Audit Page" roles="Auditors" />

    <siteMapNode url="Administrator\AddUser.aspx" title="Add User"
roles="Admins" />
    <siteMapNode url="Administrator\AlterRole.aspx" title="Alter Role"
roles="Admins" />
    </siteMapNode>

</siteMap>
```

And now when I run the application again, you'll notice that the "Audit" and "Approver" links are hidden from the TreeView (since I'm still not in those roles):

The "administrator" link is still available, however, because I am a member of the "Admins" role.

Please download project source code from www.rmb.com.np and install into your computer and have experience all.

5 Bibliography

- http://weblogs.asp.net
- http://www.asp.net/
- http://msdn.microsoft.com