ASP.NET MVC framework - Security

Some [posts ago](http://blog.maartenballiauw.be/post/2007/12/ASPNET-MVC-Framework---Basic-sample-application.aspx), I started playing with the ASP.NET MVC framework. In an example I'm creating, I'm trying to add Forms-based security.

"Classic" ASP.NET offers a nice and easy way to set security on different pages in a web application, trough Web.config. In the example I'm building, I wanted to allow access to "/Entry/Delete/" only to users with the role "Administrator". So I gave the following snip a try:

<location path="/Entry/Delete">
   <system.web>
     <authorization>
       <allow roles="Administrators"/>
       <deny users="\*"/>
     </authorization>
   </system.web>
</location>

This seems to work in some occasions, but not always. Second, I think it is very confusing to define security in a different place than my route table... Since the ASP.NET MVC framework is built around "dynamically" changing URL schemes, I'm not planning to maintain my Web.config security for each change...

In an ideal world, you would specify permissions for a route at the same location as you specify the route. Since the ASP.NET MVC framework is still an [early CTP](http://weblogs.asp.net/scottgu/archive/2007/12/09/asp-net-3-5-extensions-ctp-preview-released.aspx), perhaps this might be added in future versions. For now, the follwing strategies can be used.

## Code Access Security

Luckily, the .NET framework offers a nice feature under the name "CAS" (Code Access Security). Sounds scary? Perhaps, but it's useful in the MVC security context!

The idea behind CAS is that you specify security requirements using attributes. For example, if authentication is required in my EntryController (serving /Entry/...), I could use the following code snippet:

[PrincipalPermission(SecurityAction.Demand, Authenticated=true)]
public class EntryController : Controller {
    // ...
}

Now let's try my example from the beginning of this post. The URL "/Entry/Delete" is routed to my EntryController's Delete method. So why not decorate that method with a CAS attribute?

[ControllerAction]
[PrincipalPermission(SecurityAction.Demand, Role="Administrator"]
public void Delete(int? id) {
   ...
}

This snippet makes sure the Delete method can only be called by users in the role "Administrator"

## Exception handling

Problem using the CAS approach is that you are presented an ugly error when a security requirement is not met. There are two possible alternatives for catching these Exceptions.

### Alternative 1: Using Global.asax

In Global.asax, you can specify an Application\_Error event handler. Within this event handler, you can catch specific types of Exceptions and route them to the right error page. The following example redirects each SecurityException to the /Login, my LoginController:

protected void Application\_Error(object sender, EventArgs e) {
    Exception ex = Server.GetLastError().GetBaseException();
    if (ex is SecurityException) {
        Response.Redirect("/Login");
    }
}

### Alternative 2: Use more attributes!

[Fredrik Normén](http://weblogs.asp.net/fredriknormen/archive/2007/11/19/asp-net-mvc-framework-exception-handling.aspx) has posted an [ExceptionHandler attribute](http://weblogs.asp.net/fredriknormen/archive/2007/11/19/asp-net-mvc-framework-exception-handling.aspx) on his blog, which allows you to specify which type of Exception should be handled by which type of view. Hope this makes it into a future ASP.NET MVC framework version too!

### Alternative 3: Use in-line CAS

Another option is to use in-line CAS. For example, you can do the folluwing in your ControllerAction:

try {
    PrincipalPermission permission = new PrincipalPermission(User.Identity.Name, "Administrators", true);
    permission.Demand();
} catch (SecurityException secEx) {
    // Handle the Exception here...
    // Redirect to Login page, for example.
}