**Security in Microsoft MVC 5 web based applications and services using OWIN Cookie based authentication and Authorization Attributes.**

Security should be a part of a .Net web site from the start; it is not an afterthought. Insufficient security can open up a company to lawsuits and compromise customer data, opening the window for identity theft.

**Authentication vs. Authorization:**

Authentication and Authorization are separate concepts. Authentication involves identifying the user is who they say they are. Authentication is the process of determining who a user is via credentials, such as username/ password and increasingly, authentication via 3rd party resources like Google, Facebook and Twitter. After authentication comes authorization. Just as your local bank might know who you are, it doesn’t give you access to the vault just because it knows you. Authorization involves what rights and possibly roles an authenticated or unauthenticated user might have, what web site data they can access and possibly edit and what services and process they can use.

**Authentication modes in MVC 5 .Net 4.5:**

With the introduction of MVC 5. Microsoft has given us 4 authentication modes: None, Individual User Accounts, Organizational accounts and Windows Authentication.

None is as it implies, no authentication or authorization at all; this mode is used for public sites that pass no secure data.

Individual User Account authorization uses Cookie Based authentication, sometimes still called Forms Authentication, with additional capabilities provided by the Owin middleware provided by MVC 5. It provides robust and customizable authentication, allowing the use of custom login screen, validating user login data via a SQL or other data stores, or potentially 3rd party authentication methods and customizable authorization via roles.

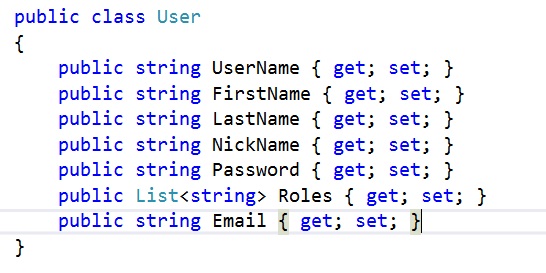
Organizational accounts use Active Directory, Windows Azure active Directory, or Office 365.

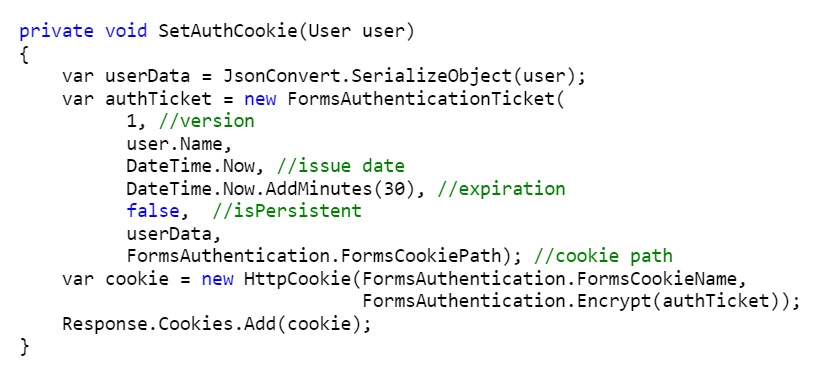
Windows Authorization is primarily intended for Intranet applications behind a company’s firewall and uses the users Windows Account Credentials. This can provide a single sign-on experience across several intranet applications.

**Forms Authentication in MVC 3:**

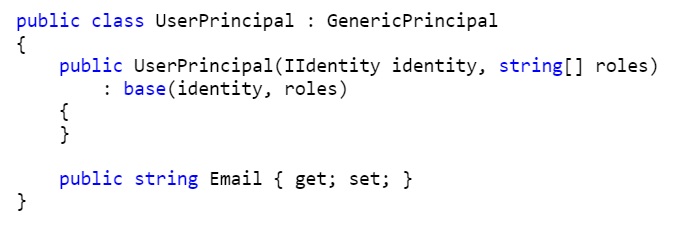
Prior to .Net 4.5, most Internet sites used Forms Authentication. To save anything more that roles and the username in the cookie, it involved building a model to represent those propeties it would be serialized and saved into the forms authentication ticket, which is then encrypted and saved into the cookie.

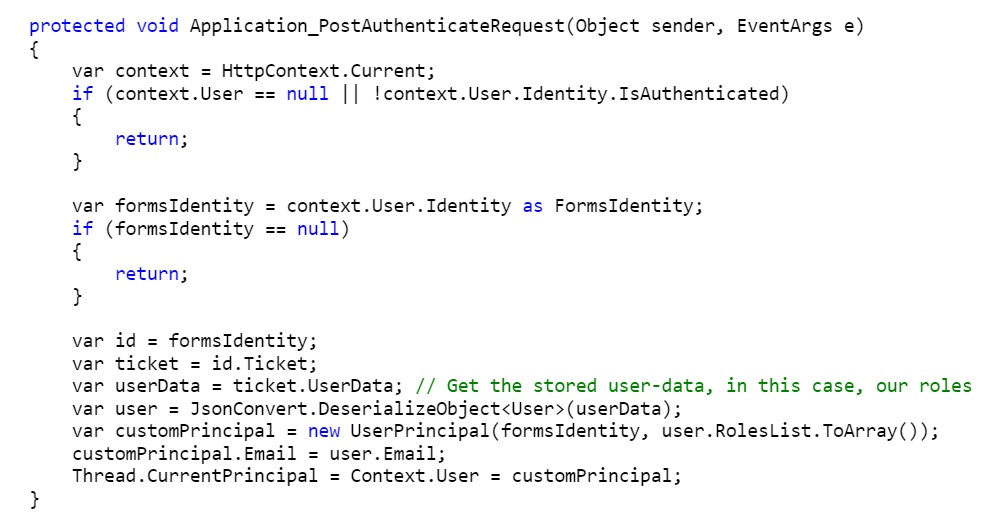
Then in the login method, after authentication of the users credentials, which can retrieve the user data, a call can be made to a method to Build and save the user’s data and roles.





When the users next request comes in, that cookie is read back to the system in the PostAuthenticationRequest where the cookie is decrypted and the custom user is pulled out. It’s data is then put into a Custom UserPrincipal object which inherits from General Principal, where it is then attached to the current thread as the current principal. This can then be used in the controllers by casting it back to a Custom Principal where it’s properties are available.





**OWIN cookie based Authentication:**

In .Net 4.5, using cookie based authentication, the login process is very similar, but the objects and processes are different.

After creating an MVC 5 application with Individual Authorization, the first noticeable change from the MVC 3 Forms authentication project is that there is no Authorization node in the

web.config.

Next, in the root directory of the application, there is a partial class Startup, decorated with an OWIN Startup Directive which causes it to be started at application start. This has a Configure method which executes a ConfigureAuth method which exists in a second partial Startup class in the app\_Start folder, it’s here that the login path and authentication method is set. When an unauthenticated user attempts to access a secured controller/method the OWIN middleware will change the 401 unauthorized status code to a 302 and redirect the user to this path, and set the “returnurl” property to the page that generated the unauthenticated request so that a user can be rerouted to that page upon authentication. This is also where you can enable 3rd party authentication by un-commenting the code that sets up the 3rd party authentication.

Controllers:

The Home controller is not decorated with an [Authorize] attribute, nor are any of its Action methods. This controller and its Action methods are open to access by an unauthenticated or authenticated user.

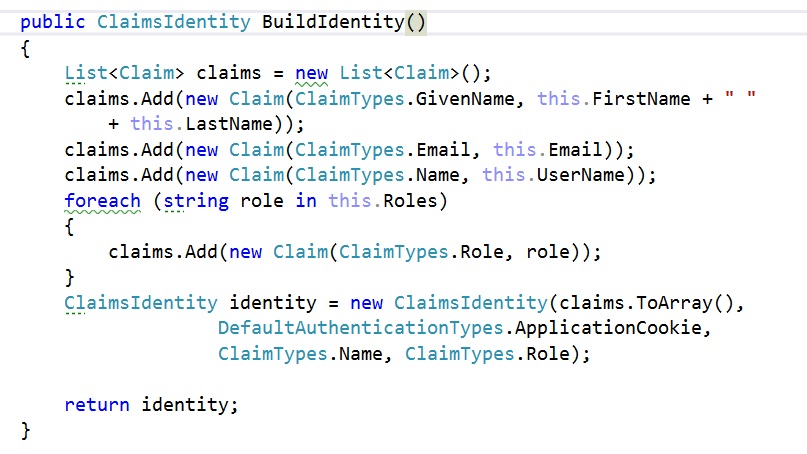
The AccountController is decorated with the [Authorize] attribute, therefore, access to its action methods are only accessible by authenticated users, except for those action methods decorated with the [AllowAnonymous] attribute, which will allow an unauthorized user to access it.

The authorize attribute:  
The authorize attribute has several modes which you can use to lock things down further than just authorized or not by specifying [Authorize(users=”[username]”) (or a comma separated list of uaers), and by using [Authorize(roles=“[role]”) (or a comma separated list of roles).

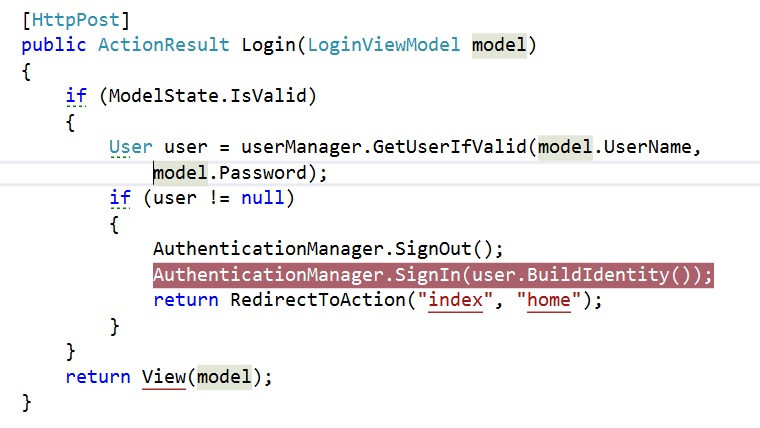
Where the OWIN cookie based authorization really shines is in what data it is able to save without customizing. In the project generated via the VS template, all the work is done via the Microsoft.AspNet.Identity’s AuthenticationManager, which will retrieve all the needed data from the SQL database, and setup the cookie and save it. This isn’t the only way though, and if you want to add more to the cookie, or want to do the work outside the SQL database, you can build your own, and again, it’s quite easy.

Create a user with the fields you need. Fields that OWIN recognizes include: Country, DateOfBirth, Email, Gender, GivenName, HomePhone , MobilePhone, Name, OtherPhone, PostalCode, StateOrProvince, StreetAddress, Surname and a List<string> of Roles.

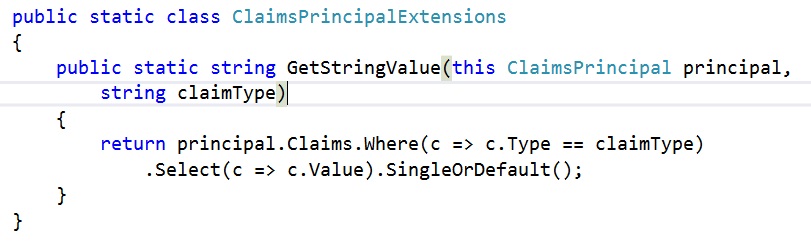
Build Claims identity object, and place the data elements into it using code like this.



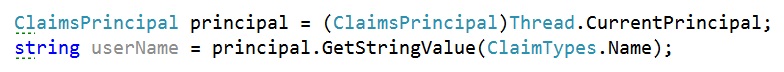
And then in the login post method:



The user data can then be accessed in a controller via the ClaimsPrincipal object, which can be accessed by casting the Thread.CurrentPrincipal to a Claims principal. Data from the user (a separate method would need to be built to access roles as it is a list) can be accessed with a simple extension method.



And in the controller:



References:

**Understanding OWIN Forms authentication in MVC** **5** <http://blogs.msdn.com/b/webdev/archive/2013/07/03/understanding-owin-forms-authentication-in-mvc-5.aspx>

## Tametoshi <http://www.cnblogs.com/neozhu/p/4519342.html>

[**Forms Authentication with claims**](http://martinwilley.com/blog/2014/03/07/FormsAuthenticationWithClaims.aspx) http://martinwilley.com/blog/CommentView,guid,f4ed2670-33c2-4ee9-9309-0b53bfcbabbf.aspx