.NET Framework 4.5+ MVC OpenID Connect example

The standard library for OpenID Connect published by the Microsoft for .NET Framework 4 is not compatible with OpenAthens Keystone. OpenAthens has published an updated library that makes it possible to connect to OpenAthens Keystone in .NET 4.5 or later. Earlier versions are not supported.

It is assumed that user has knowledge of developing applications using ASP.NET in Visual Studio. This example is based on the ASP.NET MVC Web Application template.

A completed sample web application created using these instructions is available at https://github.com/openathens/KeystoneConnectorDotNet4Sample - to use this:

1. Check out the above repository
2. Create a new Application record in the OpenAthens Service Provider dashboard, using https://localhost:44328 as the Application URL (with no trailing /). Copy the same URL (including the port number) to the Redirect URL (including trailing /).
3. Update the appSettings in Web.config using the Client Id and Client Secret of the application record created above.

Goal in this example

Authenticate a user and display all the received claims on a page. In the real world you would read the claims and feed them into your authorisation / user-session management process.

Instructions

Create project

Create a new project in Visual Studio, selecting the template "ASP.NET Web Application (.NET Framework)" - select framework version 4.5 or higher. At the second stage, select the MVC template option. Don’t select any authentication options at this stage.

Configure the website to run on https by default:

1. In the solution explorer, select the project folder, then in the properties pane, change SSL Enabled to true. Copy the SSL URL that is created.
2. Right click the project and select properties, and in the Web tab, paste the https copied above into the Project Uri field in place of the existing http URL.

Create a new Application record in the OpenAthens Service Provider dashboard, using the URL you copied in step 1 as the Application URL (with no trailing /). Copy the same URL (including the port number) to the Redirect URL.

Add the dependencies

Use NuGet package manager to add the following packages:

Microsoft.AspNet.Identity.Owin
Microsoft.Owin.Host.SystemWeb
OpenAthens.Owin.Security.OpenIdConnect

Configuration

Add the following appSettings to Web.config:

```xml
<add key="oidc:Authority" value="https://connect.openathens.net" />
<add key="oidc:ClientId" value="{clientId from the OIDC application record created in the publisher dashboard}" />
<add key="oidc:ClientSecret" value="{clientSecret from the OIDC application created in the publisher dashboard}" />
<add key="oidc:RedirectUrl" value="{redirectUrl set in the publisher dashboard}" />
```

For added security to avoid accidentally checking secure strings into source control, you could place these settings in a separate config file outside the project root and include a reference to, e.g.:
Add a Startup class that initialises the OpenID Connect authentication via an Owin assembly binding. Following the convention used in other samples from Microsoft, this is a partial class split across two files. Firstly, Startup.cs in the project root:

```csharp
using Microsoft.Owin;
using Owin;
[assembly: OwinStartup(typeof(WebApplication1.Startup))]
namespace WebApplication1
{
    public partial class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            ConfigureAuth(app);
        }
    }
}
```

and secondly, Startup.Auth.cs in the App_Start folder. (you’ll need to remove ".App_Start" from the namespace in the created file so it matches the namespace in Startup.cs):

```csharp
using Microsoft.IdentityModel.Protocols.OpenIdConnect;
using Microsoft.Owin.Security;
using Owin;
using System.Configuration;
namespace WebApplication1
{
    public partial class Startup
    {
        public static string OidcAuthority = ConfigurationManager.AppSettings["oidc:Authority"];
        public static string OidcRedirectUrl = ConfigurationManager.AppSettings["oidc:RedirectUrl"];
        public static string OidcClientId = ConfigurationManager.AppSettings["oidc:ClientId"];
        public static string OidcClientSecret = ConfigurationManager.AppSettings["oidc:ClientSecret"];
        public void ConfigureAuth(IAppBuilder app)
        {
            app.SetDefaultSignInAsAuthenticationType(CookieAuthenticationDefaults.AuthenticationType);
            app.UseCookieAuthentication(new CookieAuthenticationOptions());
            var oidcOptions = new OpenIdConnectAuthenticationOptions
            {
                Authority = OidcAuthority,
                ClientId = OidcClientId,
                ClientSecret = OidcClientSecret,
                GetClaimsFromUserInfoEndpoint = true,
                PostLogoutRedirectUri = OidcRedirectUrl,
                RedirectUri = OidcRedirectUrl,
                ResponseType = OpenIdConnectResponseType.Code,
                Scope = OpenIdConnectScope.OpenId
            };
            app.UseOpenIdConnectAuthentication(oidcOptions);
        }
    }
}
```

Configure global security settings
Add settings to Application_Start in Global.asax.cs for the anti-forgery token identifier. The ServicePointManager line is not needed if you're using version 4.6 or later:

```csharp
using System.IdentityModel.Claims;
using System.Web.Helpers;
...
protected void Application_Start()
{
    ...
    AntiForgeryConfig.UniqueClaimTypeIdentifier = ClaimTypes.NameIdentifier;
    ServicePointManager.SecurityProtocol |= SecurityProtocolType.Tls12;
}
...
```

Add login and claims links:

Add the _LoginPartial.cshtml to the Views\Shared folder, to display a Login link if a user session does not exist and will display links to logout and view the returned claims if a user session does:

```csharp
@if (Request.IsAuthenticated)
{
    using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id = "logoutForm", @class = "navbar-right" })))
    {
        @Html.AntiForgeryToken()
        <li>
        </li>
        <li><a href="javascript:document.getElementById('logoutForm').submit()">Log off</a></li>
    }
}
else
{
    <ul class="nav navbar-nav navbar-right">
        <li>@Html.ActionLink("Log in", "Login", "Account", routeValues: null, htmlAttributes: new { id = "loginLink" })</li>
    </ul>
}
```

Reference this in _Layout.cshtml after the main nav bar:

```html
...<ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("About", "About", "Home")</li>
    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
</ul>
@Html.Partial("_LoginPartial")
...```

Create claims page

Create a new folder in Views called Account. Add a new partial view called Claims and paste in the following code which will display all claims that have been returned:

```html
...<ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("About", "About", "Home")</li>
    <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
</ul>
@Html.Partial("_LoginPartial")
...```
@using System.Security.Claims

@{
    ViewBag.Title = "Claims";
}

<h2>@ViewBag.Title</h2>
<h4>Claims Present in the Claims Identity:</h4>
<table class="table table-striped table-bordered table-hover table-condensed claim-table">
    <tr>
        <th class="claim-type claim-data claim-head">Claim Type</th>
        <th class="claim-data claim-head">Claim Value Type</th>
        <th class="claim-data claim-head">Claim Value</th>
    </tr>
    @foreach (Claim claim in ClaimsPrincipal.Current.Claims)
    {
        <tr>
            <td class="claim-type claim-data">@claim.Type</td>
            <td class="claim-data">@claim.ValueType</td>
            <td class="claim-data">@claim.Value</td>
        </tr>
    }
</table>

Create AccountController.cs in the Controllers folder:

```csharp
using System.Linq;
using System.Web;
using System.Web.Mvc;
namespace WebApplication1.Controllers
{
    public class AccountController : Controller
    {
        // GET: Login
        public void Login(string returnUrl = "/")
        {
            if (!Request.IsAuthenticated)
            {
                return;
            }
            Response.Redirect("/");
        }
        public void LogOff()
        {
            if (Request.IsAuthenticated)
            {
                var authTypes = HttpContext.GetOwinContext().Authentication.GetAuthenticationTypes();
                HttpContext.GetOwinContext().Authentication.SignOut(authTypes.Select(t => t.AuthenticationType).ToArray());
            }
            Response.Redirect("/");
        }
        [Authorize]
        public ActionResult Claims()
        {
            return View();
        }
    }
}
```

Example of accessing the attributes

An example of accessing the OpenAthens scoped affiliation claim is:

```csharp
var personScopedAffiliation = (User.Identity as ClaimsIdentity)?.Claims.FirstOrDefault(c => c.Type == "eduPersonScopedAffiliation")?.Value;
```
You can then use this to make authorisation decisions.