Experimenting with ASP.NET Core Authentication Schemes

Wednesday, January 2, 2019

Some software is easier to understand if you remove the software from it's usual environment and try some experiments. ASP.NET Security components, for example. What is the impact of having multiple authentication schemes? Why does a `ClaimsPrincipal` have multiple identities? What does it mean to `SignOutAsync` on an `HttpContext`?

You'll never use the following code in a real application. But, you might use this code to tinker and experiment.

First, we'll setup two cookie authentication schemes during `ConfigureServices` – cookie1 and cookie2.

```
services.AddAuthentication(options =>
{
    options.DefaultScheme = "cookie1";
})
.AddCookie("cookie1", "cookie1", options =>
{
    options.Cookie.Name = "cookie1";
    options.LoginPath = "/loginc1";
 })
.AddCookie("cookie2", "cookie2", options =>
{
    options.Cookie.Name = "cookie2";
    options.LoginPath = "/loginc2";
});
```

Next, we'll add some middleware that allows for identity sign-in and sign-out without getting bogged down in password validations.

```
app.Use(next =>
{
    return async ctx =>
    {
        switch(ctx.Request.Path)
```

```
        {
            case "/loginc1":
                var identity1 = new ClaimsIdentity("cookie1");
                identity1.AddClaim(new Claim("name", "Alice-c1"));
                await ctx.SignInAsync("cookie1", new ClaimsPrincipal(identity1));
                break;
            case "/loginc2":
                var identity2 = new ClaimsIdentity("cookie2");
                identity2.AddClaim(new Claim("name", "Alice-c2"));
                await ctx.SignInAsync("cookie2", new ClaimsPrincipal(identity2));
                break;
            case "/logoutc1":
                await ctx.SignOutAsync("cookie1");
                break;
            case "/logoutc2":
                await ctx.SignOutAsync("cookie2");
                break;
            default:
                await next(ctx);
                break;
        }
    };
});

app.UseAuthentication();
```

Now it's time for the experiments. What happens when trying to reach pages or controllers with the following attributes?

- `[Authorize]`
- `[Authorize(AuthenticationSchemes ="cookie1")]`
- `[Authorize(AuthenticationSchemes ="cookie2")]`
- `[Authorize(AuthenticationSchemes ="cookie1, cookie2")]`

When visiting those resources, it's educational to dump out what we know about the user given the authorize conditions, and how the output changes if we change the default auth scheme.

```
<h2>User</h2>
@foreach (var identity in User.Identities)
{
    <div>Authentication Type: @identity.AuthenticationType</div>
    <table class="table">
```

```
        @foreach (var claim in identity.Claims)
        {
            <tr>
                <td>@claim.Type</td>
                <td>@claim.Value</td>
            </tr>
        }
    </table>
}
```

I've also found it useful, even in real applications, to have a page that dumps out information about the available authentication schemes. Quite often the setup is obscured by helpful extension methods we use inside of `ConfigureServices`. A page model like the following will grab the information.

```
public class AuthDumpModel : PageModel
{
    private readonly AuthenticationService authenticationService;

    public AuthDumpModel(IAuthenticationService authenticationService)
    {
        this.authenticationService = (AuthenticationService)authenticationService;
    }

    public IEnumerable<AuthenticationScheme> Schemes { get; set; }
    public AuthenticationScheme DefaultAuthenticate { get; set; }
    public AuthenticationScheme DefaultChallenge { get; set; }
    public AuthenticationScheme DefaultForbid { get; set; }
    public AuthenticationScheme DefaultSignIn { get; set; }
    public AuthenticationScheme DefaultSignOut { get; set; }

    public async Task OnGet()
    {
        Schemes = await authenticationService.Schemes.GetAllSchemesAsync();
        DefaultAuthenticate = await authenticationService.Schemes.GetDefaultAuthenticateSchemeAsync();
        DefaultChallenge = await authenticationService.Schemes.GetDefaultChallengeSchemeAsync();
        DefaultForbid = await authenticationService.Schemes.GetDefaultForbidSchemeAsync();
        DefaultSignIn = await authenticationService.Schemes.GetDefaultSignInSchemeAsync();
        DefaultSignOut = await authenticationService.Schemes.GetDefaultSignOutSchemeAsync();
    }
}
```

And now we can see what's installed, and where the defaults lead.

```
<h2>Auth Schemes</h2>

<table class="table">
    <tr>
        <th>DisplayName</th>
        <th>Name</th>
        <th>Type</th>
    </tr>
    @foreach (var scheme in Model.Schemes)
    {
        <tr>
            <td>@scheme.DisplayName</td>
            <td>@scheme.Name</td>
            <td>@scheme.HandlerType</td>
        </tr>
    }
</table>
<div>DefaultAuthenticate : @Model.DefaultAuthenticate.Name</div>
<div>DefaultForbid: @Model.DefaultForbid.Name</div>
<div>DefaultSignIn: @Model.DefaultSignIn.Name</div>
<div>DefaultSignOut: @Model.DefaultSignOut.Name</div>
```

OdeToCode by K. Scott Allen