Integrating SRAM OIDC Authentication in a C# Blazor Application

Introduction

This guide provides a comprehensive walk-through to integrate OpenID Connect (OIDC) authentication with SRAM in a C# application. We will create a Blazor Server application using Visual Studio (which has a Blazor Server boiler template) and C#.
By the end of the guide, you'll achieve the following:

- Prepare your SRAM account, create a new collaboration (read more here), and register your service.
- In Visual Studio, create a C# Blazor application with OIDC authentication for SRAM or enhance an existing application to support it.
- Understand the conceptual use of authentication and authorization in C# and Blazor using OIDC.
- Ensure seamless user login through SRAM using OIDC.

Prerequisites

- Visual Studio (Latest version with .NET and web development workload installed
- An account and collaboration on SRAM.
- A SRAM service (application/platform) registered with your collaboration. You can request a service after you have set-up your collaboration at Surf via this form.
- Client ID (APP-ID) and Client Secret from SRAM your service registration.
- Callback URL registered in your SRAM service.

Setting up SRAM

Before diving into the integration process, ensure you have properly set up SRAM:

- **Register with SRAM**: If you haven't already, create an account on SRAM. This account provides you with the necessary credentials and management tools.

- **Register your service**: After creating an account, register your service (application/platform) with SRAM. During this process, specify callback URLs, desired scopes, and other configurations.

- **Configure callback URL with SRAM**: Once you have decided on a callback URL (typically https://yourdomain.com/oidc_callback), you need to submit it to SRAM, often via a form in the SRAM dashboard or portal. This ensures that SRAM knows where to redirect users after they have been authenticated. It's always a good idea to include a localhost URL in there for testing the solution locally.

- **Integrate with SRAM**: With the service registered, use the provided credentials (client ID and client secret) and endpoint details to prepare for your service's authentication flow with SRAM.

Step-by-step Implementation:

Create a new Blazor Server project:

- Open **Visual Studio**.
- Click on '**Create a new project'**.
- In the 'Create a new project' dialog, search for 'Blazor' and choose the **Blazor Server App** template.
- Click **Next**.
- Name your project, choose a location, and click **Next**.
- Choose Framework. For Authentication Type, set it to 'None'. We're choosing 'None' at this point because we'll be implementing OIDC authentication with SRAM in the subsequent steps. This ensures that our project starts without any predefined authentication mechanism, allowing us to integrate SRAM without any conflicts or overlapping configurations.
- Click **Create**.

Install the necessary package:

- In the **Solution Explorer**, right-click on your project name and select **Manage Nuget Packages..**

- Click on the **Browse** tab, search for '**Microsoft.AspNetCore.Authentication.OpenIdConnect'**, and install the package.

Set up OIDC in Program.cs:

In the Solution Explorer, navigate to 'Program.cs'. Insert the OIDC configuration code snippet after the '**builder'** initialization and before the '**app'** creation.

This configuration will set up OpenID Connect (OIDC) authentication for your application:

```
builder.Services
       .AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
       .AddCookie(OpenIdConnectDefaults.AuthenticationScheme)
       .AddOpenIdConnect("oidc", options =>
       {
           options.Authority = "https://proxy.acc.sram.eduteams.org/";
           options.ClientId = "Your-EntityId-Goes-Here";
           options.ClientSecret = "Your-ClientSecret-Goes-Here";
           options.CallbackPath = "/oidc_callback";
           options.ResponseType = "code";
           options.GetClaimsFromUserInfoEndpoint = true;
           options.Scope.Add("openid");
           options.ClaimActions.MapJsonKey(ClaimTypes.Name, "name");
           options.Events.OnRedirectToIdentityProvider = context =>
           {
               context.ProtocolMessage.RedirectUri = "https://localhost:7275/oidc_callback";
               return Task.CompletedTask;
           };
       });
```

After inserting the code, your application will be configured to use OpenID Connect (OIDC) for authentication through the SRAM provider. The configuration includes:

- Specifying the SRAM authority, which is responsible for authenticating users and issuing tokens.
- Setting up your app's credentials with the ClientId and ClientSecret to ensure secure communication between your app and SRAM.
- Defining a callback path, which is the route SRAM will redirect to after a user has been authenticated.
- Using the authorization code flow, indicated by the ResponseType set to "code".
- Fetching additional user details after authentication from SRAM's user info endpoint.
- Requesting the "openid" scope, which is essential for OIDC authentication.
- Mapping certain claims from SRAM to your application, in this case, the user's name.
- Specifying the redirect URI for local testing purposes. Ensure this matches the redirect URI registered with SRAM.

By implementing this configuration, your application will be able to securely authenticate users via SRAM, receive user information, and ensure that communication between your app and SRAM is secure and trusted.

Note: Remember to replace 'Your-EntityId-Goes-Here' with the APP-ID obtained from your SRAM environment and 'Your-ClientSecret-Goes-Here' with your Client Secret. Also, the 'RedirectUri'' specified in the example above needs to match the callback url you have set in your SRAM service.
It's crucial to store these values securely and avoid hardcoding them directly into your application's source code, especially for production environments.

Integrate Authentication:

Continue in the Program.cs file. After the app.UseRouting(); line, insert the following code:

```
app.UseAuthentication();
app.UseAuthorization();
```

Add the Login Page:

- In the Solution Explorer, right-click on your **Pages** folder.
- Select **Add > New Item**.
- In the "Add New Item" dialog, search for "**Razor Page**" and select it.
- Name the new page 'Login.cshtml' and click **Add**.

Replace the content of 'Login.cshtml' with the following code:

```
@page
@model SRAM.OIDC.Blazor_server.Pages.LoginModel
@{
}
```

Create a new class in the Pages folder and name it **Login.cshtml.cs** and insert the following code:

```
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace SRAM.OIDC.Blazor_server.Pages
{
    public class LoginModel : PageModel
    {
        public async Task OnGet(string redirectUri)
        {
            await HttpContext.ChallengeAsync("oidc", new
                AuthenticationProperties
            { RedirectUri = redirectUri });
        }
    }
}
```

Add the Callback Page

- Follow the same steps as above, but this time, name the new page **Callback.cshtml**.

Replace the content of 'Callback.cshtml' with:

```
@page "oidc_callback"
@model SRAM.OIDC.Blazor_server.Pages.CallbackModel
@{
}
```

Create a new class in the Pages folder and name it 'Callback.cshtml.cs' and add the following code:

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace SRAM.OIDC.Blazor_server.Pages
{
    public class CallbackModel : PageModel
    {
        public void OnGet()
        {
        }
    }
}
```

The Login page serves as an entry point to start the OIDC authentication process, while the Callback page acts as a landing point after the user has been authenticated by the OIDC provider. It's important to structure these pages correctly, as they play crucial roles in the OIDC authentication flow.

Add the Home (Index) Page.

- As previously explained, create a new page in the Pages folder called Index.cshtml (if it doesn't already exist).
- Replace the contents with the following:

```
@page "/"
@inject AuthenticationStateProvider AuthProvider

<PageTitle>Index</PageTitle>

<h1>SRAM Blazor application</h1>

<p>This app demonstrates how to connect to SRAM using OIDC.</p>

<AuthorizeView>
    <Authorized>
        <p>Welcome to your new app, @context.User.Identity?.Name!</p>
    </Authorized>
    <NotAuthorized>
        <div>
            <a href="login?redirectUri=/">Log in</a>
        </div>
    </NotAuthorized>
</AuthorizeView>
```

This page acts as the main landing page for your application and displays content depending on the user's authentication status. Users who aren't authenticated will be provided with a link to **log in** using our newly created OIDC authentication with SRAM.

Run your application.

- Save all changes.
- Press F5 or click the green Start button in Visual Studio to run your application.

Additional Steps

These steps are not required for authentication but can be beneficial in your application.

Mapping Attributes from SRAM

SRAM has many attributes originating from the underlying research or educational institute. A comprehensive list can be found here. We can use the ones in the column OIDC claim. To access the groups defined in the collaboration, based on the table, we need the field eduperson_entitlement in the eduperson_entitlement scope. We can achieve this by modifying the OIDC configuration in Program.cs. Begin by requesting the scope, then map the returned field to a claim. Add the following code below where we specified the “openid” scope “name” claim:

```
options.Scope.Add("eduperson_entitlement");
options.ClaimActions.MapJsonKey(ClaimTypes.Role, "eduperson_entitlement");
```

Authentication in Code-behind and Entire Pages

You've already seen how to use AuthorizeView to adjust content on pages based on authentication status. An alternative is to hide the full code behind (for instance, in an API) or an entire page. In classes, use the [Authorize] attribute above a class or method. For razor pages and components, use @attribute [Authorize].

Role-Based Authentication with SRAM Roles

Beyond the foundational steps, you can also utilize role-based authentication. Simply add the attribute ‘[Authorize(Roles ="Admin")]’ above a class or method. The attribute's usage is well-documented by Microsoft. Given that SRAM uses a lengthy naming convention, like 'urn:mace:surf.nl:sram:group:collaboration-name:sub-collaboration-name:app-admin', it's practical to define it as a Policy in Program.cs:

```
builder.Services.AddAuthorizationCore(options =>
{
     options.AddPolicy("RequireAdminRole", policy =>
     policy.RequireRole('urn:mace:surf.nl:sram:group:collaboration-name:sub-collaboration-name:app-
admin'));
 });
```

Now, use the identifier RequireAdminRole in both AuthorizeView and Controller as follows:

```
<AuthorizeView Policy="RequireAdminRole"></AuthorizeView>
[Authorize(Policy ="RequireAdminRole")]
```

Conclusion

By following the above steps, you have successfully set up OIDC authentication in a Blazor web application using Visual Studio and C#. Always ensure the confidentiality of your Client ID (APP-ID) and especially your Client Secret to maintain the security of your application.

Note: The configuration provided here is just a guideline to help you understand the essential components of setting up OIDC authentication with SRAM. It's crucial to understand that this is an example, and you should tailor the configuration to match your application's specific requirements and security considerations.