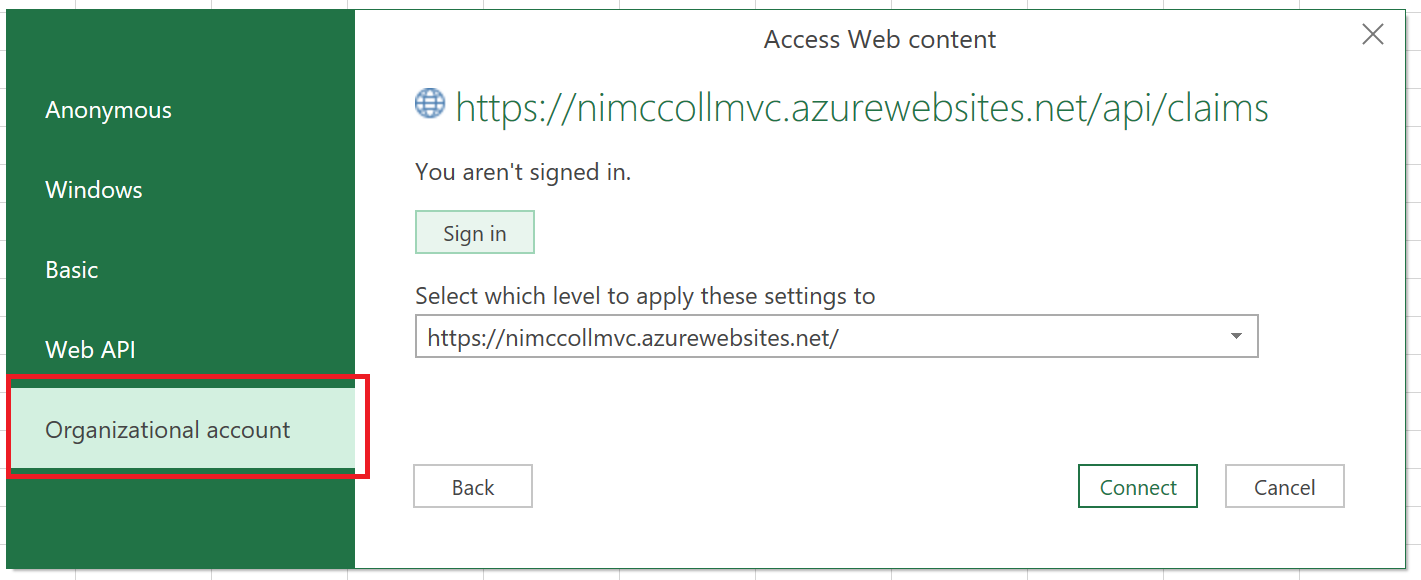
**Overview**

Excel Power Query and Power BI expect a specific WWW-Authenticate HTTP header to be returned in the 401 unauthorized response from a REST API when the “Organizational account” option is selected from the Access Web Content dialog box.



The WWW-Authenticate header must be in the following format:

WWW-Authenticate authorization\_uri=”{your tenant login URL}”

Here is an example taken from a PostMan session.



When Excel Power Query or Power BI receive a 401 unauthorized response from the REST API and encounter this specific WWW-Authenticate header, they will redirect to the specified login URL to allow the user to sign-in and obtain an access token for the REST API.

**Configuring your ASP.Net or ASP.Net Core WebAPI to support Excel Power Query and Power BI**

There are two things that are essential to making your ASP.Net WebAPI or ASP.Net Core WebAPI accessible from Excel Power Query and Power BI. First, you must return the additional WWW-Authenticate header above when your WebAPI returns a 401 unauthorized response.

*ASP.Net*

In an ASP.Net application this can be accomplished by adding the following code to the Application\_EndRequest event handler of the Global.asax.cs file as follows.

protected void Application\_EndRequest()

{

if (HttpContext.Current.Response.StatusCode == 401)

{

// Add a WWW-Authenticate header so the client application knows where to redirect to for authorization

string authorizationUri = ConfigurationManager.AppSettings["ida:AuthorizationUri"];

HttpContext.Current.Response.AddHeader("WWW-Authenticate", $"authorization\_uri=\"{authorizationUri}\"");

}

}

Your web.config file should contain an app setting with an appropriate value for ida:AuthorizationUri. A sample set of app settings entries for an ASP.Net WebAPI is shown below.

<add key="ida:Tenant" value="{your Azure AD tenant name here}" />

<add key="ida:Audience" value="{your APIs Azure AD application URI here}" />

<add key="ida:Issuer" value="https://sts.windows.net/{your Azure AD tenant ID here}/" />

<add key="ida:AuthorizationUri" value="https://login.microsoftonline.com/{your Azure AD tenant name here}" />

The OWIN startup class of the ASP.Net WebAPI should be configured to use the WindowsAzureActiveDirectoryBearerAuthentication middleware as follows.

public void ConfigureAuth(IAppBuilder app)

{

WindowsAzureActiveDirectoryBearerAuthenticationOptions options = new WindowsAzureActiveDirectoryBearerAuthenticationOptions();

options.Tenant = ConfigurationManager.AppSettings["ida:Tenant"];

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateAudience = true,

ValidAudience = ConfigurationManager.AppSettings["ida:Audience"],

ValidateIssuer = true,

ValidIssuer = ConfigurationManager.AppSettings["ida:Issuer"]

};

app.UseWindowsAzureActiveDirectoryBearerAuthentication(options);

}

*ASP.Net Core*

For an ASP.Net Core application you must leverage the events of the authentication middleware to insert the WWW-Authenticate header into the 401 unauthorized response. Typically, we would leverage the Azure AD Bearer Authentication middleware to protect our Azure AD secured REST API, however, this middleware does not support the events necessary to insert the WWW-Authenticate header into the response. Therefore, we will leverage the more generic JWT Bearer Authentication middleware and configure it to communicate with Azure AD. This will allow us to tap into the OnChallenge event and add the necessary WWW-Authenticate header to the 401 unauthorized response. Sample code taken from the ConfigureServices method of the Startup.cs file is shown below.

public void ConfigureServices(IServiceCollection services)

{

string stsDiscoveryEndpoint = Configuration.GetValue<string>("STSDiscoveryEndpoint");

string audience = Configuration.GetValue<string>("Audience");

string authorizationUri = Configuration.GetValue<string>("AuthorizationUri");

ConfigurationManager <OpenIdConnectConfiguration> configManager = new ConfigurationManager<OpenIdConnectConfiguration>(stsDiscoveryEndpoint, new OpenIdConnectConfigurationRetriever());

OpenIdConnectConfiguration config = configManager.GetConfigurationAsync().Result;

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters()

{

ValidateIssuer = true,

ValidIssuer = config.Issuer,

ValidateAudience = true,

ValidAudience = audience,

IssuerSigningKeys = config.SigningKeys

};

options.Events = new JwtBearerEvents()

{

OnChallenge = (context) =>

{

// Add a WWW-Authenticate header so the client application knows where to redirect to for authorization

context.HttpContext.Response.Headers.Add("WWW-Authenticate", $"authorization\_uri=\"{authorizationUri}\"");

return Task.FromResult(0);

}

};

});

services.AddControllers();

}

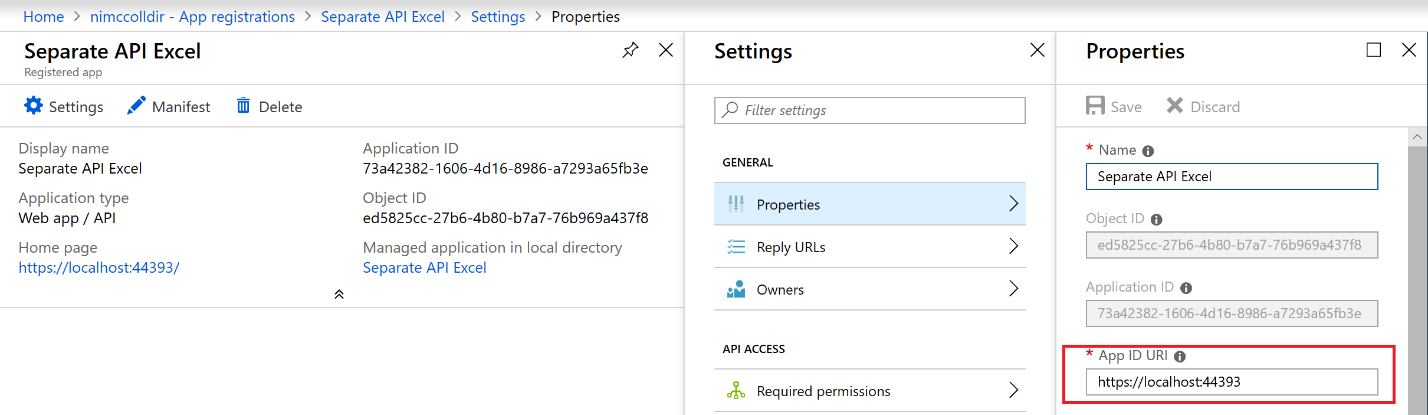
Your appsettings.json file should contain settings with appropriate values for STSDiscoveryEndpoint, Audience, and AuthorizationUri. A sample set of entries from an appsettings.json file is shown below.

"STSDiscoveryEndpoint": "https://login.microsoftonline.com/{your Azure AD tenant name here}/.well-known/openid-configuration",

"Audience": "{your APIs Azure AD application URI here}",

"AuthorizationUri": "https://login.microsoftonline.com/{your Azure AD tenant name here}",

The second thing you must do in order for your WebAPI to support Excel Power Query and Power BI is to modify the App ID Uri of your application registration in Azure Active Directory. The App ID Uri must be the scheme (http or https), host name, and port of the URL of your WebAPI. Because Excel Power Query and Power BI are completely unaware of the App ID Uri of your application in Azure AD, they always send the “authority” of the URL which is the combination of scheme, host name, and port. If your App ID Uri does not match this value, you will receive an error that a registered application could not be found in Azure AD. An example Azure AD application registration with an appropriate value is shown below:



Once these two items are in place, you will be able to successfully authenticate and use your ASP.Net WebAPI from Excel Power Query or Power BI.

Complete sample code for an ASP.Net WebAPI and an ASP.Net Core WebAPI can be found in my GitHub repository located at <https://github.com/nimccoll/AADSecuredRESTAPI>.