Work with claims-based identities

07/21/2017 • 5 minutes to read • Contributors

In this article

Sample code

Claims in Azure AD

When a user signs in, Azure AD sends an ID token that contains a set of claims about the user. A claim is simply a piece of information, expressed as a key/value pair. For example, `email=bob@contoso.com`. Claims have an issuer — in this case, Azure AD — which is the entity that authenticates the user and creates the claims. You trust the claims because you trust the issuer. (Conversely, if you don't trust the issuer, don't trust the claims!)

At a high level:

1. The user authenticates.
2. The IDP sends a set of claims.
3. The app normalizes or augments the claims (optional).
4. The app uses the claims to make authorization decisions.

In OpenID Connect, the set of claims that you get is controlled by the scope parameter of the authentication request. However, Azure AD issues a limited set of claims through OpenID Connect; see Supported Token and Claim Types. If you want more information about the user, you'll need to use the Azure AD Graph API.

Here are some of the claims from Azure AD that an app might typically care about:

Claim type in ID token	Description
aud	Who the token was issued for. This will be the application's client ID. Generally, you shouldn't need to worry about this claim, because the middleware automatically validates it. Example: `"91464657-d17a-4327-91f3-2ed99386406f"`
groups	A list of Azure AD groups of which the user is a member. Example: `["93e8f556-8661-4955-87b6-890bc043c30f", "fc781505-18ef-4a31-a7d5-7d931d7b857e"]`
iss	The issuer of the OIDC token. Example: `https://sts.windows.net/b9bd2162-77ac-4fb2-8254-5c36e9c0a9c4/`
name	The user's display name. Example: `"Alice A."`
oid	The object identifier for the user in Azure AD. This value is the immutable and non-reusable identifier of the user. Use this value, not email, as a unique identifier for users; email addresses can change. If you use the Azure AD Graph API in your app, object ID is that value used to query profile information. Example: `"59f9d2dc-995a-4ddf-915e-b3bb314a7fa4"`
roles	A list of app roles for the user. Example: `["SurveyCreator"]`

Claim type in ID token	Description
tid	Tenant ID. This value is a unique identifier for the tenant in Azure AD. Example: `"b9bd2162-77ac-4fb2-8254-5c36e9c0a9c4"`
unique_name	A human readable display name of the user. Example: `"alice@contoso.com"`
upn	User principal name. Example: `"alice@contoso.com"`

This table lists the claim types as they appear in the ID token. In ASP.NET Core, the OpenID Connect middleware converts some of the claim types when it populates the Claims collection for the user principal:

- oid > `http://schemas.microsoft.com/identity/claims/objectidentifier`
- tid > `http://schemas.microsoft.com/identity/claims/tenantid`
- unique_name > `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name`
- upn > `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn`

Claims transformations

During the authentication flow, you might want to modify the claims that you get from the IDP. In ASP.NET Core, you can perform claims transformation inside of the **AuthenticationValidated** event from the OpenID Connect middleware. (See Authentication events.)

Any claims that you add during **AuthenticationValidated** are stored in the session authentication cookie. They don't get pushed back to Azure AD.

Here are some examples of claims transformation:

- **Claims normalization**, or making claims consistent across users. This is particularly relevant if you are getting claims from multiple IDPs, which might use different claim types for similar information. For example, Azure AD sends a "upn" claim that contains the user's email. Other IDPs might send an "email" claim. The following code converts the "upn" claim into an "email" claim:

  C#　　Copy

  ```
  var email = principal.FindFirst(ClaimTypes.Upn)?.Value;
  if (!string.IsNullOrWhiteSpace(email))
  {
      identity.AddClaim(new Claim(ClaimTypes.Email, email));
  }
  ```

- Add **default claim values** for claims that aren't present — for example, assigning a user to a default role. In some cases this can simplify authorization logic.

- Add **custom claim types** with application-specific information about the user. For example, you might store some information about the user in a database. You could add a custom claim with this information to the authentication ticket. The claim is stored in a cookie, so you only need to get it from the database once per login session. On the other hand, you also want to avoid creating excessively large cookies, so you need to consider the trade-off between cookie size versus database lookups.

After the authentication flow is complete, the claims are available in `HttpContext.User`. At that point, you should treat them as a read-only collection—for example, using them to make authorization decisions.

Issuer validation

In OpenID Connect, the issuer claim ("iss") identifies the IDP that issued the ID token. Part of the OIDC authentication flow is to verify that the issuer claim matches the actual issuer. The OIDC middleware handles this for you.

In Azure AD, the issuer value is unique per AD tenant (`https://sts.windows.net/<tenantID>`). Therefore, an application should do an additional check, to make sure the issuer represents a tenant that is allowed to sign in to the app.

For a single-tenant application, you can just check that the issuer is your own tenant. In fact, the OIDC middleware does this automatically by default. In a multitenant app, you need to allow for multiple issuers, corresponding to the different tenants. Here is a general approach to use:

- In the OIDC middleware options, set **ValidateIssuer** to false. This turns off the automatic check.
- When a tenant signs up, store the tenant and the issuer in your user DB.
- Whenever a user signs in, look up the issuer in the database. If the issuer isn't found, it means that tenant hasn't signed up. You can redirect them to a sign up page.
- You could also blacklist certain tenants; for example, for customers that didn't pay their subscription.

For a more detailed discussion, see Sign-up and tenant onboarding in a multitenant application.

Using claims for authorization

With claims, a user's identity is no longer a monolithic entity. For example, a user might have an email address, phone number, birthday, gender, etc. Maybe the user's IDP stores all of this information. But when you authenticate the user, you'll typically get a subset of these as claims. In this model, the user's identity is simply a bundle of claims. When you make authorization decisions about a user, you will look for particular sets of claims. In other words, the question "Can user X perform action Y" ultimately becomes "Does user X have claim Z".

Here are some basic patterns for checking claims.

- To check that the user has a particular claim with a particular value:

  C#	Copy

  ```csharp
  if (User.HasClaim(ClaimTypes.Role, "Admin")) { ... }
  ```

  This code checks whether the user has a Role claim with the value "Admin". It correctly handles the case where the user has no Role claim or multiple Role claims.

  The **ClaimTypes** class defines constants for commonly used claim types. However, you can use any string value for the claim type.

- To get a single value for a claim type, when you expect there to be at most one value:

  C#	Copy

  ```csharp
  string email = User.FindFirst(ClaimTypes.Email)?.Value;
  ```

- To get all the values for a claim type:

  C#	Copy

  ```csharp
  IEnumerable<Claim> groups = User.FindAll("groups");
  ```

For more information, see Role-based and resource-based authorization in multitenant applications.

Next