Introduction to OpenID Connect

In this lesson we will learn about OpenID Connect, its mechanics and how it complements the OAuth 2.0 protocol.

What is OpenID Connect?

What is OpenID Connect? Don't worry, it's not an entire new security protocol to learn. OpenID Connect is just a simple identity layer on top of the OAuth 2.0 protocol that allows clients to verify the identity of the End-User.

Notice the emphasis on "verify the identity". Identity is the one piece that OAuth 2.0 misses because, think about it, OAuth 2.0 allows an application to get access to resources on behalf of a user, but the client application never really knows who is that end-user that provided authorization.

Let's understand why this is so important.

OAuth 2.0 is not enough

Let's look at our simplified OAuth 2.0 flow once again. Just as before, John uses the client application to request authorization to retrieve his inventory bag, owned by the Inventory microservice. The Identity microservice receives this request, authenticates John via the sign in form and, after the usual OAuth protocol exchanges we learned about in the previous lesson, it sends back an access token to John's client.

Now, imagine that John doesn't use this token in his client app, but instead he shares it with Jane, by just pasting it in an email or an instant message. Jane then somehow provides this token to her own copy of the client app and then uses it to request John's inventory bag. The inventory microservice will find no issues with the token, so it will gladly return the items in John's inventory to Jane's client.

Everything looks good from the point of view of our microservices, but the client app was able to get access to protected resources when it shouldn't. This is because access tokens are not meant for the client app, but only for the protected resources. The client app can't even understand the access tokens. It needs a way to verify that access to the resources was granted to whoever is making use of the app.

Let's see how OpenID Connect can help with this problem.

The OpenID Connect authentication flow

Since OpenID Connect is based on OAuth 2.0, the involved participants are mostly the same, but they receive slightly different names:

- **The end-user** is the person, and in this case an actual human, that needs to get authenticated.

- **The relying party** is the client application that outsources end-user authentication to an OpenID Provider.

- **The OpenID Provider** is the server capable of authenticating the End-User and provide claims to a relying party about the user.

The flow starts the same way as with the OAuth 2.0 flow, but a key difference is that in the authentication request a very specific scope called “openid” is required. This is really the only thing that signals to the OpenID Provider that the relying party needs authentication. Other optional standard scopes could also be used here to retrieve additional claims about the end-user.

Upon receiving this request, the OpenID Provider redirects the end user to the sign in page as usual, optionally asks for consent to specific permissions, and then returns an authorization code. And, just as before, the relying party then uses the authorization code to request tokens from the OpenID Provider.

Another big difference here is that the OpenID Provider returns not one but at least two tokens, the access token and the ID token. This additional ID token is what the relying party can then use to verify the identity of the authenticated user. It contains one or more claims about the End-User, depending on the requested scopes. A key claim here is the sub claim, which provides the unique identifier of the user.

After this the relying party could either request more information about the end-user via the OpenId Provider UserInfo endpoint, or it can just use the access token to make requests to any of the resource servers, like Inventory in this example.

What OpenID Connect adds

Like I mentioned before, the main benefit of using OpenID Connect on top of OAuth 2.0 is the fact that the client can now verify the identity of the end-user. But there are several other benefits of using this protocol:

- **Standard scopes.** Scopes in OAuth 2.0 are very loose, so each service has to define them according to their needs. In OpenID connect there is a standard set of scopes you can use to request additional claims about the user like profile, email, address and phone.
- **Standard claims.** There are several standard claims that the OpenId Provider can return upon request like the required sub claim, which provides the unique identifier of the user, but also many other claims like given name, nickname, email, birthdate, and many others.
- **User info endpoint.** This is a standard endpoint in the OpenID provider that can be used to retrieve more information about the end user that might not be available as part of the ID token.
- **Single sign-on.** More than a specific OpenID Connect offering, single sign on is a side benefit since a single open id provider can be used to authenticate end-users coming from different apps and in most cases the user will be required to perform the authentication only the first time. This happens for instance when after authenticating in Gmail you are automatically authenticated to YouTube, AdSense, Google Analytics and other Google apps.

- **Configuration discovery.** OpenId providers define a well-known endpoint that apps can use to discover the several other endpoints and features they can use as part of the user authentication flow. We'll explore this endpoint in detail in a future lesson.
- **Certified OpenID providers.** OpenID providers can get certified by the OpenID foundation via standard suite of tests and requirements. There are dozens of certified OpenID providers you can choose from today giving you lots of flexibility to use the one that better suites the needs of your apps.

In the next lesson we will learn about IdentityServer, the OpenID Connect provider we will use through this course.