Tokens, OAuth2 and JWT

00:01: Welcome to the third module of course 6, "Tokens, OAuth2, and JWT." This module is all about tokens, and more specifically, it's about JSON Web Tokens. First, we're going to do a quick pass over the history of different token implementations. We're then going to have a quick look at the high level structure of a token and basically work on understanding what a token is, why we need a token, and how it actually makes things better. And finally, we're going to settle on JSON Web Tokens. And we're going to first see why that is actually better than the old token implementations. And then, we're going to work on actually setting that up with in Spring Security OAuth. Let's now start talking about tokens and let's have a quick look at the history of the tokens as it applies to various security mechanisms. And we will start with SAML out of the WS Star space. This was very, very heavily XML based, its cryptographic options were off the chart so it had a lot of signing and encryption options available. But it also needed a pretty advanced XML stack, and that's of course something that not every device is going to have.

01:11: So basically, as we started to go more into the mobile space, that XML stack simply did not exist on mobile. And so, as a direct consequence of all of that complexity, there was a joint venture between Microsoft, Google, and Yahoo, and the simple web token standard was created. Now, it turns out that this standard is actually a little bit too simple so they've had not enough cryptographic options that is, for example, just an option for symmetric signing and so, there's no asymmetric signing in this case. And so, it was just too simple and too restrictive. So the next step was the creation of the JSON Web Token Standard. And this new standard basically combines what was good from both of those previous approaches so it has a lot more cryptographic options but it also is a lot simpler than SAML. And more importantly, it doesn't rely on XML. So, of course, using JSON is much more widely supported and so JWT definitely hits a sweet spot. And it has seen a quick and very, very impressive adoption.

02:12: So let's have a quick look at how the token actually looks like. A very simple high level look at the token structure is it has three sections, and those sections are separated by a dot. So you have a header section, you have the payload section, and then you have the signature. And we're going to have a look at all of those sections. First the header section, we have here a quick example of how that will look like in practice. We have the metadata here and we have some information here about the algorithms that we used. And so, this will be Base64 encoded and this will represent the first out of the three parts of the token structure. Next, we have the payload. And the payload basically represents the claims that this token puts forward so we have two types of claims. We have reserved claims, and we have application specific claims. So beyond the reserved claims like issuer or expiration or audience or things like that, we also have the ability to provide our own application specific claims, and that is of course well supported by the standard, and it also really pays off when we marry JWT with OAuth because we will make use of some of those custom claims.

03:26: And finally, we have the token signature. The signature is of course the part that guarantees the integrity of the Token and essentially guarantees that the token has not been tampered with. And the signature in this case will contain the header information, the payload information, both of which are encrypted with a secret. So in a nutshell, this is basically how the JWT Token is structured. Now that we have an initial understanding of how JWT works, let's look at the implementation and let's actually start moving the authorization server to being capable of issuing JWT Tokens.

04:02: So the first thing we're going to switch is the token store and we're going to use the JWT specific style of token store here. And once we implement that, we will see what the next step is because we'll see that the JWT store actually requires something else. So we can see here that the JWT Token Store actually requires a token converter. So this is a converter that basically decodes and encodes the JWT Token into OAuth information, so basically bridges the gap between JWT as a token, having all of this information encoded inside the token, and what OAuth actually requires. So let's define that Bean. Let's define this converter which, by the way, doubles as an enhancer, as you can see here, so let's define that.

[pause]

05:06: Okay, so we defined the AccessTokenConverter and we used that converter to define our new JWT Token store. Also a quick side note here is that we're using symmetrical cryptography, we're using the signing key to sign our tokens. And when we're going to reach the resource server configuration, which is next, we're going to see how we actually need to define the exact same JWT AccessTokenConverter. And more importantly, we'll need to use the exact same sign-in key, of course, in order for the resource server to be able to consume and to check the tokens that the authorization server will issue. So this is really an important point to understand here. Now, in this particular project, we're not really going to have to do a lot because the authorization server and the resource server are actually living in the same Spring context and so we are sharing this Bean. However if the authorization server would be a different project, and that is a common way to set it up, then we would have had to define this JWT AccessTokenConverter exactly the same both in the authorization server, as well as in the resource server.

06:16: So definitely an important thing to understand, first why we would need to do that, and second why we're not gonna really do that here and again this is just because we are sharing this Bean. And of course this is a very hard-coded implementation. A good step here would really be to pull the sign-in key into a property. So let's do that quickly.

[pause]

06:48: Okay, so we are injecting the sign-in key, via the value annotation. And we are providing this default value, but if we ever want to configure this externally now we can. The final configuration of the authorization server is going to be down here. So we are basically going into the end points configuration here, and we are just pointing this to the same token converter that we used before. And that really about wraps us up for the authorization server configuration. The next step is going to be the resource server.

07:22: Here is our resource server configuration, but this part is going to be really really quick and the reason for that is that we really don't have to do anything here. Because we are using a simple project, and so the resource server and the authorization server are essentially living within the same Spring context, and within the same project, we don't really need to define anything new here. As I was just saying the only thing that we would have had to define if these two were separate, would have been the converter. But since that Bean will apply here as well we don't really need to do that, and the resource server will be perfectly fine understanding JWT Tokens signed by the authorization server.

08:02: So let's now actually start everything and let's run the test. And as you can see everything passes and we are all good. The switch to JWT as our token format is fully complete. One final thing that I want to go over just to show the exact format of the JWT Token here, is we will hit the endpoint of the authorization server, and will get an access token back with JWT running now. So let's see how that actually looks.

08:36: Okay, so we can now see the exact format of the JWT Token we see the header section here, we then see the payload section up until here and then we see the signature. So everything looks good, we are fully transitioned to using JWT Tokens, tests are passing and we are able to generate a token by hitting the API.

09:00: Okay, so what are the take-aways of this module? Well, first we now have some context around different token based security solutions. We had a look at a few of them and we started to understand how the token is structured, and how it's protected. And of course finally we settled on JWT as our token solution and we set that up with Spring Security OAuth, we discussed the two solutions to sign the token and we saw how it actually works in practice.

09:27: Okay, hope you're excited, see you in the next one.