JAX-RS Token Authorization

JAX-RS: Token Authorization

- Introduction
- Backwards compatibility configuration note
- Maven dependencies
- Claims based access control
  - Claims annotations
  - Enforcing Claims authorization
- Role based access control
  - SimpleAuthorizingInterceptor
  - Using annotations

Introduction

CXF JAX-RS offers an extension letting users to enforce a new fine-grained Claims Based Access Control (CBAC) based on Claim and Claims annotations as well as ClaimMode enum class. It works with SAML tokens and with JWT tokens (from the 3.3.0 release onwards).

See also JAX-RS XML Security, JAX-RS SAML and JAX-RS JOSE.

Backwards compatibility configuration note

From Apache CXF 3.1.0, the WS-Security based configuration tags used to configure XML Signature or Encryption ("ws-security-") have been changed to just start with "security-". Apart from this they are exactly the same. Older "ws-security-", values continue to be accepted in CXF 3.1.0. To use any of the configuration examples in this page with an older version of CXF, simply add a "ws-" prefix to the configuration tag.

The package for Claim, Claims and ClaimMode annotations has changed from "org.apache.cxf.rs.security.saml.authorization" to "org.apache.cxf.security.claims.authorization". Starting from CXF 2.7.1, the default name format for claims is "urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified" instead of "http://schemas.xmlsoap.org/ws/2005/05/identity/claims".

From the 3.3.0 release, the Claims access control annotations/interceptors now work with JWT tokens (as well as SAML tokens). This resulted in the following package changes:

- ClaimsAuthorizingInterceptor has moved from the cxf-rt-security-saml module to the cxf-rt-security module. The package name of the ClaimsAuthorizingInterceptor has changed: from org.apache.cxf.rs.security.saml.interceptor.ClaimsAuthorizingInterceptor to org.apache.cxf.security.claims.interceptor.ClaimsAuthorizingInterceptor.
- ClaimsAuthorizingFilter has moved from the cxf-rt-security.xml module to the cxf-rt-frontend-jaxrs module. The package name of the ClaimsAuthorizingFilter has changed: from org.apache.cxf.rs.security.saml.authorization.ClaimsAuthorizingFilter to org.apache.cxf.jaxrs.security.ClaimsAuthorizingFilter.

Maven dependencies

```xml
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-security</artifactId>
  <version>3.3.0</version>
</dependency>
```

In addition, cxf-rt-rs-security-xml is required if you are working with SAML tokens, and cxf-rt-rs-security-jose-jaxrs is required if you are working with JWT tokens.

Claims based access control

Claims annotations

Here is a simple code fragment to secure a service object using Claims annotations:
import org.apache.cxf.security.claims.authorization.Claim;
import org.apache.cxf.security.claims.authorization.Claims;

@Path("/bookstore")
public class SecureClaimBookStore {

    @POST
    @Path("/books")
    @Produces("application/xml")
    @Consumes("application/xml")
    @Claims({
        @Claim({"admin" }),
        @Claim(name = "http://claims/authentication-format",
            format = "http://claims/authentication",
            value = {"fingertip", "smartcard" })
    })
    public Book addBook(Book book) {
        return book;
    }
}

SecureClaimBookStore.addBook(Book) can only be invoked if Subject meets the following requirement: it needs to have a Claim with a value "admin" and another Claim confirming that it got authenticated using either a 'fingertip' or 'smartcard' method. Note that @Claim("admin") has no name and format classifiers set - it relies on default name and format values, namely "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" and "urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified" ("http://schemas.xmlsoap.org/ws/2005/05/identity/claims" before CXF 2.7.1) respectively. These default values may change in the future depending on which claims are found to be used most often - but as you can see you can always provide name and format values which will scope a given claim value.

Note that in the above example, a Claim with the name "http://claims/authentication-format" has two values, 'fingertip' and 'smartcard'. By default, in order to meet this Claim, Subject needs to have a Claim which has either a 'fingertip' or 'smartcard' value. If it is expected that Subject needs to have a Claim which has both 'fingertip' and 'smartcard' values, then the following change needs to be done:

import org.apache.cxf.security.claims.authorization.Claim;
import org.apache.cxf.security.claims.authorization.Claims;

@Path("/bookstore")
public class SecureClaimBookStore {

    @POST
    @Path("/books")
    @Produces("application/xml")
    @Consumes("application/xml")
    @Claims({
        @Claim({"admin" }),
        @Claim(name = "http://claims/authentication-format",
            format = "http://claims/authentication",
            value = {"fingertip", "smartcard" },
            matchAll = true)
    })
    public Book addBook(Book book) {
        return book;
    }
}

Claims can be specified using individual @Claim annotation, they can be set at the class level and overridden at the method level and finally a lax mode of check can be specified:
import org.apache.cxf.security.claims.authorization.Claim;
import org.apache.cxf.security.claims.authorization.Claims;

@Path("/bookstore")
@Claim({"user"})
public class SecureClaimBookStore {

    @POST
    @Path("/books")
    @Produces("application/xml")
    @Consumes("application/xml")
    @Claims({
        @Claim({"admin"}),
        @Claim(name = "http://claims/authentication-format",
                format = "http://claims/authentication",
                value = {"fingertip", "smartcard"},
                matchAll = true)
    })
    public Book addBook(Book book) {
        return book;
    }

    @GET
    @Claim(name = "http://claims/authentication-format",
            format = "http://claims/authentication",
            value = {"password"},
            mode = ClaimMode.LAX)
    public Book getBook() {
        //...
    }

    @GET
    public BookList getBookList() {
        //...
    }
}

In the above example, getBookList() can be invoked if Subject has a Claim with the value "user"; addBook() has it overridden - "admin" is expected and the authentication format Claim too; getBook() can be invoked if Subject has a Claim with the value "user" and it also must have the authentication format Claim with the value "password" - or no such Claim at all.

org.apache.cxf.rt.security.claims.interceptor.ClaimsAuthorizingInterceptor (org.apache.cxf.rt.security.saml.interceptor.ClaimsAuthorizingInterceptor before CXF 3.3.0) enforces the CBAC rules. This filter can be overridden and configured with the rules directly which can be useful if no Claim-related annotations are expected in the code. Map nameAliases and formatAliases properties are supported to make @Claim annotations look a bit simpler, for example:

```java
@Claim(name = "auth-format", format = "authentication", value = {"password"})
```

where "auth-format" and "authentication" are aliases for "http://claims/authentication-format" and "http://claims/authentication" respectively.

### Enforcing Claims authorization

Simply adding Claims annotations are per the examples above is not sufficient to enforce claims based authorization.

First we need to configure the appropriate interceptors/filters to authenticate the type of token we are interested in extracting claims from. See the JAX-RS SAML page for information on how to configure SAML, and the JAX-RS JOSE page for information on how to configure JWT.

For both SAML and JWT, once the incoming token is validated, a ClaimsSecurityContext security context will be created containing the claims contained in the token, as well as the authenticated subject and role (claims).

To enforce claims authorization, a ClaimsAuthorizingInterceptor must be set as an "inInterceptor", passing it a reference to the secured object. There is also a JAX-RS filter wrapper around ClaimsAuthorizingInterceptor available, which is called ClaimsAuthorizingFilter.

An instance of org.apache.cxf.rs.security.saml.authorization.ClaimsAuthorizingFilter (note org.apache.cxf.rs.security.claims.ClaimsAuthorizingFilter from CXF 3.3.0) is used to enforce CBAC. It's a simple JAX-RS filter wrapper around ClaimsAuthorizingInterceptor.
Here is an example of enforcing Claims authorization against a JWT token. BookStoreAuthn is the service object which is annotated with Claims annotations. The ClaimsAuthorizingFilter is added as a JAX-RS provider to the endpoint, wrapping the serviceBean. A JwtAuthenticationFilter instance is also added to validate the received JWT token and to set up the ClaimsSecurityContext. The rs.security.signature.in.properties property is used to verify the signature on the received token.

```xml
<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.jose.jwt.BookStoreAuthn"/>

<bean id="claimsHandler" class="org.apache.cxf.jaxrs.security.ClaimsAuthorizingFilter">
    <property name="securedObject" ref="serviceBean"/>
</bean>

<bean id="jwtAuthzFilter" class="org.apache.cxf.rs.security.jose.jaxrs.JwtAuthenticationFilter">
    <property name="roleClaim" value="role"/>
</bean>

<jaxrs:server address="https://localhost:${testutil.ports.jaxrs-jwt-authn-authz}/signedjwtauthz">
    <jaxrs:serviceBeans>
        <ref bean="serviceBean"/>
    </jaxrs:serviceBeans>
    <jaxrs:providers>
        <ref bean="jwtAuthzFilter"/>
        <ref bean="claimsHandler"/>
    </jaxrs:providers>
    <jaxrs:properties>
        <entry key="rs.security.signature.in.properties" value="org/apache/cxf/systest/jaxrs/security/bob.jwk.properties"/>
    </jaxrs:properties>
</jaxrs:server>
```

### Role based access control

If we have a SAML Assertion or JWT token with claims that are known to represent roles, then making those claims work with an RBAC system can be achieved easily.

**SimpleAuthorizingInterceptor**

One option is to enforce that only users in a given role can access a method in the service bean is to use CXF's SimpleAuthorizingInterceptor. It has a "methodRolesMap" property can maps method names to roles. This interceptor must then be added to the inInterceptor chain of the service endpoint. For example:
<bean id="serviceBean" class="org.apache.cxf.systest.jaxrs.security.jose.jwt.BookStoreAuthn"/>

<bean id="jwtAuthzFilter" class="org.apache.cxf.rs.security.jose.jaxrs.JwtAuthenticationFilter">
  <property name="roleClaim" value="role"/>
</bean>

<bean id="authorizationInterceptor" class="org.apache.cxf.interceptor.security.SimpleAuthorizingInterceptor">
  <property name="methodRolesMap">
    <map>
      <entry key="echoBook" value="boss"/>
      <entry key="echoBook2" value="boss"/>
    </map>
  </property>
</bean>

<jaxrs:server address="https://localhost:${testutil.ports.jaxrs-jwt-authn-authz}/signedjwtauthz">
  <jaxrs:serviceBeans>
    <ref bean="serviceBean"/>
  </jaxrs:serviceBeans>
  <jaxrs:providers>
    <ref bean="jwtAuthzFilter"/>
  </jaxrs:providers>
  <jaxrs:inInterceptors>
    <ref bean="authorizationInterceptor"/>
  </jaxrs:inInterceptors>
  <jaxrs:properties>
    <entry key="rs.security.signature.in.properties" value="org/apache/cxf/systest/jaxrs/security/bob.jwk.properties"/>
  </jaxrs:properties>
</jaxrs:server>

Using annotations

Instead of mapping method names to roles using the SimpleAuthorizingInterceptor, we can instead annotate them in the service bean with javax.annotation.security.RolesAllowed or org.springframework.security.annotation.Secured annotations. For example:

```java
import org.springframework.security.annotation.Secured;

@Path("/bookstore")
public class SecureBookStore {

  @POST
  @Secured("admin")
  public Book addBook(Book book) {
    return book;
  }
}
```

where @Secured can be replaced with @RolledAllowed if needed, the following configuration will do it:
That is all what is needed. Note that in order to help the default SAML SecurityContextProvider figure out which claims are roles, one can set the two properties as shown above - this not needed if it's known that claims identifying roles have NameFormat and Name values with the default values, which are "http://schemas.xmlsoap.org/ws/2005/05/identity/claims" and "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/role" respectively at the moment.

Note that you can have RBAC and CBAC combined for a more sophisticated access control rules be enforced while still keeping the existing code relying on @RolesAllowed or @Secured intact. Override ClaimsAuthorizingFilter and configure it with the Claims rules directly and register it alongside SimpleAuthorizingFilter and here you go.

Also note how SecureAnnotationsInterceptor can handle different types of role annotations, with @RolesAllowed being supported by default.