WildFly Elytron Security

Bearer Token Authorization

I managed to implement Bearer Token Authorization with Keycloak and Elytron in WildFly 26 in order to control access to RESTful Web services in a Web Module (.war) of an Enterprise Application (.ear), but the solution is not without problems. This is what I did:

Define an elytron token realm

```
/subsystem=elytron/token-realm=xyz2ap112-token-realm/:add(\
    principal-claim=preferred_username,\
    oauth2-introspection={\
        client-id=xyz2ap112-web-api,\
        client-secret=${env.keycloak_client_secret},\
        introspection-url=${env.keycloak_introspection_url}\
    }\
)
```

Define an elytron role decoder

```
/subsystem=elytron/simple-role-decoder=xyz2ap112-realm-access-roles/:add(\
    attribute=realm_access_roles\
)
```

Warning: the default "Token Claim Name" for Keycloak realms is "realm_access.roles". For this role decoder to work, I had to change it to "realm_access_roles" (no dot). I'll mention this again when I talk about the problems with this solution.

Define an elytron security domain

```
/subsystem=elytron/security-domain=xyz2ap112-token-security-domain/:add(\
    realms=[{realm="xyz2ap112-token-realm",role-decoder="xyz2ap112-realm-access-roles"}],\
    default-realm=xyz2ap112-token-realm,\
    permission-mapper=default-permission-mapper\
)
```

Define an elytron HTTP authentication factory

```
/subsystem=elytron/http-authentication-factory=xyz2ap112-web-api-authentication-factory/:add(\
    security-domain=xyz2ap112-token-security-domain,\
    mechanism-configurations=[{\
        mechanism-name=BEARER_TOKEN,\
        mechanism-realm-configurations=[realm-name=xyz2ap112-token-realm]\
    }],\
    http-server-mechanism-factory=global\
)
```

Define two application security domains

ejb3 subsystem

```
/subsystem=ejb3/application-security-domain=xyz2ap112-web-api-security-domain/:add(\
    security-domain=xyz2ap112-token-security-domain\
)
```

Warning: the war that contains the web services does not contain the EJBs it needs; those are in a separate EJB Module (.jar). I guess that is why I had to define this application security domain in the ejb3 subsystem.

undertow subsystem

```
/subsystem=undertow/application-security-domain=xyz2ap112-web-api-security-domain/:add(\
    http-authentication-factory=xyz2ap112-web-api-authentication-factory,\
    override-deployment-config=true\
)
```

Configure application’s jboss-web.xml and web.xml

```
<jboss-web>
    <context-root>/xyz2ap112-web-api</context-root>
    <resource-ref>
        <res-ref-name>jdbc/xyz2ap112</res-ref-name> <!-- Logical name only. -->
        <jndi-name>java:/jdbc/xyz2ap112</jndi-name> <!-- Real JNDI name. -->
    </resource-ref>
    <security-domain>xyz2ap112-web-api-security-domain</security-domain>
</jboss-web>
```

The security-domain is the application security domain defined in the undertow subsystem.

```
    <login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>xyz2ap112-web-api-security-domain</realm-name>
    </login-config>
```

The real-name in login-config is the application security domain defined in the undertow subsystem.

PROBLEMS

As I said before, this solution is not without problems. Given that my Enterprise Application (.ear) also has another Web Module (.war), which contains the GUI components of the application and no Web Services, this solution works as long as the auth-method of that second Web Module is FORM or BASIC. And, as you might have already guessed, I want to use OIDC.

Using OIDC to control access to the application is very straightforward, as properly explained by Farah Juma in her article Securing WildFly Apps with OpenID Connect. But that works as long as the "Token Claim Name" of the Keycloak realm is "realm_access.roles" (its default value). With that name the simple-role-decoder doesn’t work. So, I guess a custom role-decoder is required. Given that my application is able to define and administer roles and role assignments on its own, instead of writing a custom role-decoder, I used a constant-role-mapper to get a single role that allows the web services to execute and check permissions using the roles defined within the application. Once again, that works as long as the auth-method of that second Web Module is FORM or BASIC; with OIDC, the web services are not executed; client gets an HTTP 500 (see below). There is no additional information in the logs of any of the running WildFly (Keycloak and application).

This is the oidc.json file of the GUI Web Module:

```
{
    "client-id": "xyz2ap112-web",
    "confidential-port": 8543,
    "principal-attribute": "preferred_username",
    "provider-url": "http://localhost:8180/auth/realms/jrcam",
    "public-client": true,
    "ssl-required": "external"
}
```

And this is the client exception:

```
Exception in thread "main" javax.ws.rs.InternalServerErrorException: HTTP 500 Internal Server Error
      at org.glassfish.jersey.client.JerseyInvocation.convertToException(JerseyInvocation.java:1098)
      at org.glassfish.jersey.client.JerseyInvocation.translate(JerseyInvocation.java:883)
      at org.glassfish.jersey.client.JerseyInvocation.lambda$invoke$1(JerseyInvocation.java:767)
      at org.glassfish.jersey.internal.Errors.process(Errors.java:316)
      at org.glassfish.jersey.internal.Errors.process(Errors.java:298)
      at org.glassfish.jersey.internal.Errors.process(Errors.java:229)
      at org.glassfish.jersey.process.internal.RequestScope.runInScope(RequestScope.java:414)
      at org.glassfish.jersey.client.JerseyInvocation.invoke(JerseyInvocation.java:765)
      at org.glassfish.jersey.client.JerseyInvocation$Builder.method(JerseyInvocation.java:428)
      at org.glassfish.jersey.client.JerseyInvocation$Builder.get(JerseyInvocation.java:324)
      at org.xyz.jax.rs.client.base.AbstractFacadeServiceClient.find(AbstractFacadeServiceClient.java:28)
      at xyz2.BarrioFacadeClient.find(BarrioFacadeClient.java:40)
      at xyz2.BarrioFacadeClient.main(BarrioFacadeClient.java:24)
```

If the auth-method of the Web Services Web module is OIDC, the response the client gets is html corresponding to the login page.

```
<html xmlns="http://www.w3.org/1999/xhtml" class="login-pf">
    ...
                    <h1 id="kc-page-title">
                        Sign in to your account
                    </h1>
    ...
</html>
```

This is the oidc.json file of the Web Services Web Module:

```
{
    "client-id": "xyz2ap112-web-api",
    "confidential-port": 8543,
    "principal-attribute": "preferred_username",
    "provider-url": "http://localhost:8180/auth/realms/jrcam",
    "ssl-required": "external",
    "bearer-only": true,
    "verify-token-audience": true,
    "realm-public-key":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAk3PD30r3SQBqnO15g/Jc5z3NFnt9HLA6QlQt2QLtxvGhLcerTD2rVWCst/4NSQev9dBscFn
wxXyAoZAqTm7w0oPzlhw1Xbqt1dpKdNjMtbJxmpqzCRLTjmNatPmoAGx+9TWOPKw1qfEwZOy9xOqnCbBeT5eGCAXci+wvt8mpNX9lpAguFxgpFtyVc0a
t35Lw3BdZ13+6Ljxu6Z+mam1tQ9mwey0ubfhV3NK0eN8jruKWrCyGw6DRbmvKFTwQa5akDbMWt3H/HaSLMXBOrBKq9He6azVL3dkbdd40drgHtI8G+AN
C1NhOPzjPtuifo9U2wHD6o8S03o35mm4xjJNcqQIDAQAB",
    "credentials": {
        "secret": "8c98045a-4640-46e7-9f68-74a289e43b7e"
    }
}
```

I hope this partial solution will help someone and also that someone can tell me how to implement a complete solution.