Overview of the TCK

The MicroProfile JWT(MP-JWT) TCK consists of tests that validate the authentication and authorization of JAX-RS application endpoints secured using an MP-JWT authentication mechanism that both authenticates and authorizes access based on an MP-JWT token passed in via the HTTP Authorization header as a bearer token. There are 3 categories of unit tests in the base required TCK:

1. Tests that validate the basic behavior of the MP-JWT JsonWebToken implementation independent of any JAX-RS application. Tests in this category are marked with the TestNG groups of "jwt" and "utils".
2. Tests that validate the authentication and authorization of JAX-RS application endpoints secured using `@RolesAllowed` annotations. This includes testing the requirements for rejecting MP-JWT tokens. Tests in this category are marked with the TestNG group "jaxrs".
3. Tests that validate the injection of `o.e.m.j.ClaimValue`, `javax.json.JsonValue` and `javax.inject.Provider` wrappers for the JsonWebToken claim values. Tests in this category are marked with the TestNG groups of "cdi", "cdi-json" and "cdi-provider".

The TestNG testsuite for these categories of tests can be found in the tck directory root as tck-base-suite.xml.

Optional Java EE Container Integration Tests

In addition to the categories of tests defined above, there are optional tests that illustrate the expected container API integration for Java EE container runtimes beyond JAX-RS. These tests are marked with the TestNG groups "ejb-optional", "jacc-optional", and "servlet-optional". An MP-JWT implementation is not required to pass these tests in order to be considered a valid implementation. The TestNG testsuite for these categories + the base categories of tests can be found in the tck directory root as tck-full-suite.xml.

TCK Resource Files and Information

The TCK includes resources for the the MP-JWT token payload content as well as test issuer public and private RSA keys. These are used by the unit tests along with the `org.eclipse.microprofile.jwt.tck.util.TokenUtils` class to generate both valid and invalid tokens. A summary of the tck/resources directory contents is:

- jwt-content1.json
- testJWTCallerPrincipal.json
- usePreferredName.json
- useSubject.json
  - Used by the `TokenUtilsTest` to validate the MP-JWT implementation under test JsonWebToken implementation.
- Token1.json
- Token2.json

    - Used by the various JAX-RS and CDI tests
- privateKey.pem
    - The test issuer private key used to sign the MP-JWT tokens generated by the unit tests.
- publicKey.pem
    - The test issuer public key that MP-JWT implementations under test use to validate the token signature.

The test issuer value used in all valid test MP-JWT tokens as the `iss` claim value is "https://server.example.com".

The test issuer public key is the tck/resources/publicKey.pem file. It is included in every test WebArchive artifact as an archive classpath resource at the location /WEB-INF/classes/publicKey.pem.

The generated test JWT has its `exp`, `iat` and `auth_time` claims are set to the current time when the token is generated, as the number of seconds from 1970-01-01T00:00:00Z UTC.

Running the MicroProfile JWT Auth TCK

The TCK is designed around a set of Arquillian based unit tests that require the MP-JWT implementation under test to provide a TCK harness artifact that provides an `org.jboss.arquillian.core.spi.LoadableExtension` that installs a `org.jboss.arquillian.container.test.spi.client.deployment.ApplicationArchiveProcessor` to augment the base `org.jboss.shrinkwrap.api.spec.WebArchive` with the implementation specific artifacts, descriptors, libraries, etc. needed for the implementation to properly deploy the test web archive.

There are base implementations of the `LoadableExtension` and `ApplicationArchiveProcessor` that can be used for strait-forward augmentation scenarios, but you can always provide your own implementations. An example of the former is: https://github.com/MicroProfileJWT/wfswarm-jwt-auth-tck-viabase

while an example of the latter is: https://github.com/MicroProfileJWT/wfswarm-jwt-auth-tck

Additionally, you must provide an implementation of the `org.eclipse.microprofile.jwt.tck.util.ITokenParser` interface using the `java.util.ServiceLoader` pattern. This is used by the TCK tests that validate the basic behavior of the MP-JWT JsonWebToken implementation independent of any container runtime deployment. An example of an `ITokenParser` implementation can be found in: TCKTokenParser.java.

Creating Your Implementation TCK Harness Artifact

As described, you need to create an artifact that bundles a LoadableExtension using a Java ServiceProvider that installs an ApplicationArchiveProcessor that augments the base TCK test web application archive with the implementation specific configuration and dependencies needed to successfully deploy and test the web application with MP-JWT authentication enabled.

An example skeleton pom.xml is shown here:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed under the Apache License, Version 2.0 (the
    "License"); you may not use this file except in compliance
    with the License.  You may obtain a copy of the License at

     http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing,
    software distributed under the License is distributed on an
    "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
    KIND, either express or implied.  See the License for the
    specific language governing permissions and limitations
    under the License.
-->
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>my.groupID</groupId>
    <artifactId>jwt-auth-tck</artifactId>
    <version>1.0-SNAPSHOT</version>
    <name>MicroProfile JWT Auth TCK Harness MyCoolMP Implementation</name>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <version.wildfly.swarm>2017.7.0</version.wildfly.swarm>
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.jboss.arquillian</groupId>
                <artifactId>arquillian-bom</artifactId>
                <version>1.1.13.Final</version>
```

```
                <scope>import</scope>
                <type>pom</type>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <!-- This is the MP-JWT TCK base extension and utility classes -->①
        <dependency>
            <groupId>org.eclipse.microprofile.jwt</groupId>
            <artifactId>microprofile-jwt-auth-tck</artifactId>
            <version>1.0-SNAPSHOT</version>
        </dependency>
        <!-- Arquillian extension SPI -->②
        <dependency>
            <groupId>org.jboss.arquillian.container</groupId>
            <artifactId>arquillian-container-spi</artifactId>
        </dependency>
        <dependency>
            <groupId>org.jboss.arquillian.container</groupId>
            <artifactId>arquillian-container-test-spi</artifactId>
        </dependency>
        <!-- You need to specify a JAX-RS client implementation as the unit
        tests make use of that API, but do not specify and implementation.
        -->③
        <dependency>
            <groupId>org.jboss.resteasy</groupId>
            <artifactId>resteasy-client</artifactId>
            <version>3.1.1.Final</version>
        </dependency>

        <!-- Specify your container runtime arquillian integration and dependencies
-->
        <dependency>④
            <groupId>MY_GROUP</groupId>
            <artifactId>arquillian-container</artifactId>
            <version>${container-version}</version>
        </dependency>
        ...
    </dependencies>

...

</project>
```

① org.eclipse.microprofile.jwt:microprofile-jwt-auth-tck is the MP-JWT artifact that contains the base `LoadableExtension` and `ApplicationArchiveProcessor` classes, as well as the `ITokenParser` harness factory interface and `TokenUtils` class.

② The 2 indicated Arquillian extension SPI dependencies provide the `LoadableExtension` and `ApplicationArchiveProcessor` interfaces and dependent classes.

③ The TCK unit tests make use of the JAX-RS client API, but do not provide an implementation, so your TCK harness artifact must specify what implementation to use. Here the Resteasy implementation is being specified.

④ Lastly, you must specify the property Arquillian container runtime that is approriate for you MP-JWT implementation, along with whatever container runtime dependencies are required.

Running Your Implementation With the TCK

Once you have built and installed your TCK harness artifact, you can run the TCK tests against it by using either the `tokens-se` or `container` profiles.

tokens-se Profile

The tokens-se profile is a basic unit test for validation of your JWTPrincipal implementation. This does not require any special test container runner. This test does require that your TCK harness artifact provide an implementation of the `org.eclipse.microprofile.jwt.tck.util.ITokenParser` interface using a service provider via the `java.util.ServiceLoader` pattern, that is, there should be a META-INF/services/org.eclipse.microprofile.jwt.tck.util.ITokenParser file in your TCK harness artifact that points to your ITokenParser implementation.

To run the unit tests associated with the tokens-se profile, run the following command from within the microprofile-jwt-auth/tck directory:

```
mvn -Ptokens-se -Dtck.container.groupId={MY_GROUP} -Dtck.container.artifactId={MY_ARTIFACT} -Dtck.container.version={MY_VERSION} test
```

where you would replace the `{MY_GROUP}`, `{MY_ARTIFACT}` and `{MY_VERSION}` with the `<groupId>⋯<groupId>`, `<artifactId>⋯</artifactId>`, and `<version>⋯</version>` respectively from your TCK harness artifact.

A concrete example is for running with the TCK harness artifiact from the https://github.com/MicroProfileJWT/wfswarm-jwt-auth-tck project is:

```
mvn -Ptokens-se -Dtck.container.groupId=org.wildfly.swarm -Dtck.container.artifactId=jwt-auth-tck -Dtck.container.version=1.0-SNAPSHOT test
```

container Profile

The container profile is a test of JAX-RS client tests that validate a JAX-RS endpoint bundled in a WebArchive deployment via your implementation. These tests require Arquillian container runtime integration to properly deploy and start your container. You typically provide this via a dependency on an arquillian container artificat, for example, Tomcat based containers might include a dependency like:

```
<dependency>
  <groupId>org.jboss.arquillian.container</groupId>
  <artifactId>arquillian-tomcat-embedded-7</artifactId>
  <version>1.0.0</version>
  <scope>test</scope>
</dependency>
```

This test of tests also require the `org.jboss.arquillian.core.spi.LoadableExtension` and `org.jboss.arquillian.container.test.spi.client.deployment.ApplicationArchiveProcessor` implementations as discussed above.

To run this set of tests, issue the following command from within the microprofile-jwt-auth/tck directory:

```
mvn -Pcontainer -Dtck.container.groupId={MY_GROUP} -Dtck.container.artifactId={MY_ARTIFACT} -Dtck.container.version={MY_VERSION} test
```

where you would replace the `{MY_GROUP}`, `{MY_ARTIFACT}` and `{MY_VERSION}` with the `<groupId>⋯<groupId>`, `<artifactId>⋯</artifactId>`, and `<version>⋯</version>` respectively from your TCK harness artifact.

A concrete example is for running with the TCK harness artifiact from the https://github.com/MicroProfileJWT/wfswarm-jwt-auth-tck project is:

```
mvn -Pcontainer -Dtck.container.groupId=org.wildfly.swarm -Dtck.container.artifactId=jwt-auth-tck -Dtck.container.version=1.0-SNAPSHOT
```

Running the TCK Tests in Your Build

You can run the TCK tests from within your TCK harness build by including the following in your pom.xml:

```
    </dependencies>
    ...
        <!-- Include the MP-JWT TCK dependencies -->
        <dependency>
            <groupId>org.eclipse.microprofile.jwt</groupId>
            <artifactId>microprofile-jwt-auth-tck</artifactId>
            <version>1.0-SNAPSHOT</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.eclipse.microprofile.jwt</groupId>
            <artifactId>microprofile-jwt-auth-tck</artifactId>
            <version>1.0-SNAPSHOT</version>
            <type>test-jar</type>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
        ...
            <!-- Run the TCK tests aginst the tck-base-suite.xml -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.20</version>
                <configuration>
                    <redirectTestOutputToFile>true</redirectTestOutputToFile>
                    <suiteXmlFiles>
                        <suiteXmlFile>tck-base-suite.xml</suiteXmlFile>
                    </suiteXmlFiles>
                    <forkCount>1</forkCount>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

and then either copy the tck-base-suite.xml file from the TCK source tree into your build root, or copy the following and paste if into a tck-base-suite.xml file in your build root:

```
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd" >
<suite name="microprofile-jwt-auth-BaseTCK" verbose="1" preserve-order="true"
configfailurepolicy="continue" >

    <!-- The required base JAX-RS and CDI based tests that all MP-JWT implementations
    must pass.
    -->
    <test name="base-tests" verbose="10">
        <groups>
            <define name="base-groups">
                <include name="arquillian" description="Arquillian internal"/>
                <include name="utils" description="Utility tests"/>
                <include name="jwt" description="Base JsonWebToken tests"/>
                <include name="jaxrs" description="JAX-RS invocation tests"/>
                <include name="cdi" description="Base CDI injection of ClaimValues"/>
                <include name="cdi-json" description="CDI injection of JSON-P
values"/>
                <include name="cdi-provider" description="CDI injection of
javax.inject.Provider values"/>
            </define>
            <define name="excludes">
                <include name="debug" description="Internal debugging tests" />
            </define>
            <run>
                <include name="base-groups" />
                <exclude name="excludes" />
            </run>
        </groups>
        <classes>
            <class name="org.eclipse.microprofile.jwt.tck.parsing.TokenValidationTest"
/>
            <class name="org.eclipse.microprofile.jwt.tck.util.TokenUtilsTest" />
            <class
name="org.eclipse.microprofile.jwt.tck.parsing.TestTokenClaimTypesTest" />
            <class
name="org.eclipse.microprofile.jwt.tck.container.jaxrs.UnsecuredPingTest" />
            <class
name="org.eclipse.microprofile.jwt.tck.container.jaxrs.ClaimValueInjectionTest" />
            <class
name="org.eclipse.microprofile.jwt.tck.container.jaxrs.JsonValueInjectionTest" />
            <class
name="org.eclipse.microprofile.jwt.tck.container.jaxrs.ProviderInjectionTest" />
            <class
name="org.eclipse.microprofile.jwt.tck.container.jaxrs.RolesAllowedTest" />
            <class
name="org.eclipse.microprofile.jwt.tck.container.jaxrs.InvalidTokenTest" />
        </classes>
    </test>
</suite>
```

You then simply run `mvn test` to run the TCK tests. An example of using this approach can be found in the https://github.com/MicroProfileJWT/wfswarm-jwt-auth-tck repo. Running

```
[wfswarm-jwt-auth-tck 1316]$ mvn -Dswarm.resolver.offline=true test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building MicroProfile JWT Auth TCK Harness WFSwarm Implementation 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ jwt-auth-tck ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] Copying 6 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ jwt-auth-tck ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ jwt-
auth-tck ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e.
build is platform dependent!
[INFO] skip non existing resourceDirectory
/Users/starksm/Dev/JBoss/Microprofile/wfswarm-jwt-auth-tck/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ jwt-auth-tck
---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.20:test (default-test) @ jwt-auth-tck ---
[INFO] No tests to run.
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running TestSuite
[INFO] Tests run: 19, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 49.95 s - in
TestSuite
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 19, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 52.805 s
[INFO] Finished at: 2017-08-23T17:23:41-07:00
[INFO] Final Memory: 30M/619M
[INFO] ------------------------------------------------------------------------
```