# Java Token Validation

## Overview

Prerequesite: Using the MobileFirst Server to authenticate external resources (../).

MobileFirst Platform provides a Java library to facilitate the authentication of external resources.
This library is provided as a JAR (**mfp-java-token-validator-8.0.0.jar**) and is available as a **maven dependency**:

```xml
<dependency>
  <groupId>com.ibm.mfp</groupId>
  <artifactId>mfp-java-token-validator</artifactId>
  <version>8.0.0</version>
</dependency>
```

This tutorial will show how to protect a simple Java Servlet `GetBalance` with a MobileFirst Platform scope `accessRestricted`.

## Instantiating the TokenValidationManager

You need to instantiate `TokenValidationManager` to be able to validate tokens:

```java
TokenValidationManager(java.net.URI authorizationURI, java.lang.String clientId, java.lang.String clientSecret);
```

- `authorizationURI`: the URI of the Authorization server, usually the MobileFirst Platform server. For example **http://localhost:9080/mfp/api**.
- `clientId`: The confidential client ID you configured in the MobileFirst Operations Console.
- `clientSecret`: The confidential client secret you configured in the MobileFirst Operations Console.

## Validating the credentials

The `validate` API will ask the authorization server to validate the authorization header:

```java
public TokenValidationResult validate(java.lang.String authorizationHeader, java.lang.String expectedScope);
```

- `authorizationHeader`: The content of the HTTP header `Authorization`. For example it could be obtained from a `HttpServletRequest` (`httpServletRequest.getHeader("Authorization")`).
- `expectedScope`: Optional. The scope to validate the token against.

You can query the resulting `TokenValidationResult` object for either an error or valid introspection data:

```
TokenValidationResult tokenValidationRes = validator.validate(authCredentials, expectedScope);
if (tokenValidationRes.getAuthenticationError() != null) {
  // Error
  AuthenticationError error = tokenValidationRes.getAuthenticationError();
  httpServletResponse.setStatus(error.getStatus());
  httpServletResponse.setHeader("WWW-Authenticate", error.getAuthenticateHeader());
} else if (tokenValidationRes.getIntrospectionData() != null) {
  // Success
  httpServletRequest.setAttribute("introspection-data", tokenValidationRes.getIntrospectionData());
  filter.doFilter(req, res);
}
```

## Introspection data

The `TokenIntrospectionData` object returned by `getIntrospectionData()` provides you with some information about the client, such as the username of the currently active user:

```
TokenIntrospectionData introspectionData = (TokenIntrospectionData) request.getAttribute("introspection-data");
String username = introspectionData.getUsername();
```

> See the JavaDoc for more APIs.

## Cache

The `TokenValidationManager` class comes with an internal cache which caches tokens and introspection data. The purpose of this is if a request with the same header is made, we don't need to go *introspect* that token against the Authorization Server. The default cache size is **50000 items**, after this capacity is reached, the oldest token is removed. Also - it is important to note that before a token is retrieved from the cache, its expiration is checked (the expiration is a field in the introspection data), and is removed if expired. This is all done internally.

Note that the constructor of `TokenValidationManager` can also accept a cacheSize (number of introspection data items) to store:

```
public TokenValidationManager(java.net.URI authorizationURI, java.lang.String clientId, java.lang.String clientSecret, long cacheSize);
```

## Protecting a simple Java Servlet

Create a simple Java Servlet called `GetBalance`, which returns a hardcoded value:

```
@WebServlet("/GetBalance")
public class GetBalance extends HttpServlet {
   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
       //Return hardcoded value
       response.getWriter().append("17364.9");
   }

}
```

Create a `javax.servlet.Filter` implementation, called `JTVFilter`, that will validate the authorization header for a given scope:

```
public class JTVFilter implements Filter {

  public static final String AUTH_HEADER = "Authorization";
  private static final String AUTHSERVER_URI = "http://localhost:9080/mfp/api"; //Set here your authentication server URI
  private static final String CLIENT_ID = "jtv"; //Set here your confidential client ID
  private static final String CLIENT_SECRET = "jtv"; //Set here your confidential client SECRET

  private TokenValidationManager validator;
  private FilterConfig filterConfig = null;

  @Override
  public void init(FilterConfig filterConfig) throws ServletException {
    URI uri = null;
    try {
      uri = new URI(AUTHSERVER_URI);
      validator = new TokenValidationManager(uri, CLIENT_ID, CLIENT_SECRET);
      this.filterConfig = filterConfig;
    } catch (Exception e1) {
      System.out.println("Error reading introspection URI");
    }
  }

  @Override
  public void doFilter(ServletRequest req, ServletResponse res, FilterChain filter) throws IOException, ServletException {
    String expectedScope = filterConfig.getInitParameter("scope");
    HttpServletRequest httpServletRequest = (HttpServletRequest) req;
    HttpServletResponse httpServletResponse = (HttpServletResponse) res;

    String authCredentials = httpServletRequest.getHeader(AUTH_HEADER);

    try {
      TokenValidationResult tokenValidationRes = validator.validate(authCredentials, expectedScope);
      if (tokenValidationRes.getAuthenticationError() != null) {
        // Error
        AuthenticationError error = tokenValidationRes.getAuthenticationError();
        httpServletResponse.setStatus(error.getStatus());
        httpServletResponse.setHeader("WWW-Authenticate", error.getAuthenticateHeader());
      } else if (tokenValidationRes.getIntrospectionData() != null) {
        // Success
        httpServletRequest.setAttribute("introspection-data", tokenValidationRes.getIntrospectionData());
        filter.doFilter(req, res);
      }
    } catch (TokenValidationException e) {
      httpServletResponse.setStatus(500);
    }
  }

}
```

In your **web.xml**, declare an instance of `JTVFilter` and pass the **scope** `accessRestricted` as a parameter:

```
<filter>
  <filter-name>accessRestricted</filter-name>
  <filter-class>com.sample.JTVFilter</filter-class>
  <init-param>
    <param-name>scope</param-name>
    <param-value>accessRestricted</param-value>
  </init-param>
</filter>
```

Then protect your servlet with the filter:

```
<filter-mapping>
  <filter-name>accessRestricted</filter-name>
  <url-pattern>/GetBalance</url-pattern>
</filter-mapping>
```

# Sample

The simple Java servlet is available for download. You can deploy the project on your favorite application server (this was tested using Tomcat).

This external resource can be used in combination with any of the security samples on any platform.

- Make sure to update the confidential client and secret.
- Deploy either of the security checks: **UserLogin** or **PinCodeAttempts**.
- Register the matching application.
- Map the scope `accessRestricted` to the security check.
- Update the client application to make the `WLResourceRequest` to your servlet URL.