

Java Token Validator

Overview

MobileFirst Foundation provides a Java library to enforce security capabilities on external resources. The Java library is provided as a JAR file (**mfp-java-token-validator-8.0.0.jar**).

This tutorial shows how to protect a simple Java Servlet, `GetBalance`, by using a scope (`accessRestricted`).

Prerequisites:

- Read the Using the MobileFirst Server to authenticate external resources (../) tutorial.
- Understanding of the MobileFirst Foundation security framework (../..).



Adding the .jar file dependency

The **mfp-java-token-validator-8.0.0.jar** file is available as a **maven dependency**:

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

Instantiating the TokenValidationManager

To be able to validate tokens, instantiate `TokenValidationManager`.

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

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

The library exposes an API that encapsulates and simplifies the interaction with the authorization server's introspection endpoint. For a detailed API reference, see the MobileFirst Java Token Validator API reference (http://www.ibm.com/support/knowledgecenter/en/SSHS8R_8.0.0/com.ibm.worklight.apiref.doc/apiref/r_mfpf_java_token_validator_api.html?view=kc).

Validating the credentials

The `validate` API method asks the authorization server to validate the authorization header:

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

- `authorizationHeader`: The content of the `Authorization` HTTP header, which is the access token. For example, it could be obtained from an `HttpServletRequest` (`httpServletRequest.getHeader("Authorization")`).
- `expectedScope`: The scope to validate the token against, for example `accessRestricted`.

You can query the resulting `TokenValidationResult` object for an error or for 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 logic here
}

```

Introspection data

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

```

httpServletRequest.setAttribute("introspection-data", tokenValidationRes.getIntrospectionData());

```

```

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

```

Cache

The `TokenValidationManager` class comes with an internal cache which caches tokens and introspection data. The purpose of the cache is to reduce the amount of token *introspections* done against the Authorization Server, if a request is made with the same header.

The default cache size is **50000 items**. After this capacity is reached, the oldest token is removed.

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

1. 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");
    }
}

```

2. Create a `javax.servlet.Filter` implementation, called `JTVFilter`, which 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 authorization 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 filterChain) 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());
                filterChain.doFilter(req, res);
            }
        } catch (TokenValidationException e) {
            httpServletResponse.setStatus(500);
        }
    }
}

```

3. In the servlet's **web.xml** file, 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

You can deploy the project on the supported application servers (Tomcat, WebSphere Application Server full profile, and WebSphere Application Server Liberty profile).

Download the simple Java servlet (<https://github.com/MobileFirst-Platform-Developer-Center/JavaTokenValidator/tree/release80>).

Sample usage

1. Make sure to update the confidential client (`../#confidential-client`) and secret values in the MobileFirst Operations Console.
2. Deploy either of the security checks: **UserLogin** (`../user-authentication/security-check/`) or **PinCodeAttempts** (`../credentials-`

validation/security-check/).

3. Register the matching application.
4. Map the `accessRestricted` scope to the security check.
5. Update the client application to make the `WLResourceRequest` to your servlet URL.

Last modified on