**Implementation issues in the use of MCP as login-mechanism**

Author: Jinki Jung, Almir Zerem, Rasmus Madsen Jensen

In this document we would like to share our experiences and knowledge on the implementation of a client application that authenticate via MCP. The experiences mostly come from PortCDM (Portable CDM) project.

When a mobile client application or a web application try to get authentication using MCP, from the current MCP login mechanism, the use of a Keycloak client which is “out-of-the-box”(please refer to <http://www.keycloak.org/documentation.html> and <https://oauth.net/2/> ) is mandatory. This assumes that service instances are already registered in MIR in order to retrieve a Keycloak client.

**Use case**

An end-user opens mobile or browse to web-application, he/she chooses MCP log-in and is re-directed to MCP broker page, on this page the user needs to choose correct identity provider (the one that the user is registered with), the user is redirected to the page where he/she finally enters his/hers credentials.

**Issue 1: Parallel authentification with both Keycloak and certificate**

The main technical issue we have had with implementing the login mechanism was to get Keycloak and certificates to work in parallel on PortCDM infrastructure. The problem was that the Keycloak extension for Wildfly overrode all https connections and required a valid Keycloak connection. This meant that certificate authentication would not work.

**Existing solution for Problem 1**

This problem was solved by adding a custom security domain in Wildfly’s XML configuration, called ‘superauth’. This security domain flags both Keycloak and certificates as sufficient authentication. As auth method in our modules web.xml files, we specify both KEYCLOAK, and our CLIENT-CERT-WRAPPER, a custom ServletExtension that sets the use of the ‘superauth’ security domain and prevents Keycloak from interfering when we use certificate authentication (please refer the example from the PortCDM project).

|  |
| --- |
| **package** se.viktoria.stm.portcdm;  **import** io.undertow.security.api.AuthenticationMode; **import** io.undertow.security.api.SecurityContext; **import** io.undertow.security.api.SecurityContextFactory; **import** io.undertow.security.idm.IdentityManager; **import** io.undertow.security.impl.ClientCertAuthenticationMechanism; **import** io.undertow.security.impl.SecurityContextImpl; **import** io.undertow.server.HttpServerExchange; **import** io.undertow.servlet.api.DeploymentInfo; **import** org.jboss.as.security.plugins.SecurityDomainContext; **import** org.jboss.security.auth.callback.JBossCallbackHandler; **import** org.jboss.security.plugins.JBossAuthenticationManager; **import** org.jboss.security.plugins.JBossAuthorizationManager; **import** org.wildfly.extension.undertow.security.JAASIdentityManagerImpl;  **import** javax.servlet.ServletContext;  **public** **class** ClientCertWrapperServletExtension **implements** io.undertow.servlet.ServletExtension {      /\*\*      \* The method adds a client certificate wrapper to prevent keycloak from interfering when using certificate based authentication.      \*/     @Override     **public** **void** handleDeployment( DeploymentInfo deploymentInfo, ServletContext servletContext ) {         deploymentInfo.addAuthenticationMechanism( **"CLIENT-CERT-WRAPPER"**, **new** ClientCertAuthenticationMechanism.Factory( **null** ) );         deploymentInfo.setSecurityContextFactory( **new** SecurityContextFactoryExt() );         //deploymentInfo.setIdentityManager( new JAASIdentityManagerImpl( new SecurityDomainContext( new JBossCachedAuthenticationManager( "superauth", new JBossCallbackHandler() ) ) ) );     }      **public** **class** SecurityContextFactoryExt **implements** SecurityContextFactory {          **private** SecurityContextFactoryExt() {         }          /\*\*          \* This methods overrides the security context used in the Client Server exchange by forcing it to be client-cert.         \*          \* @param exchange          \* @param mode          \* @param identityManager          \* @param programmaticMechName          \* @return A security context specific to client certificate          \*/         **public** SecurityContext createSecurityContext( HttpServerExchange exchange, AuthenticationMode mode, IdentityManager identityManager, String programmaticMechName ) {             SecurityDomainContext sdc = **new** SecurityDomainContext( **new** JBossAuthenticationManager( **"superauth"**, **new** JBossCallbackHandler() ) );             sdc.setAuthorizationManager( **new** JBossAuthorizationManager( **"superauth"** ) );             SecurityContext sc = **new** SecurityContextImpl( exchange, **new** JAASIdentityManagerImpl( sdc ) );             **return** sc;         }     } } |

Once this was solved, we have been able to successfully test Keycloak authentication from mobile application Portable CDM (developed by RISE) and web-application PACT (developed by the STM project partner HiQ). These applications are used in all PortCDM ports for the validation purposes by many actors, but currently Gothenburg is the one port that we demonstrated.

**Issue 2: Premature time-out**

Default time-out is set for just only few minutes for the security manner. But this was a huge disturbance in the operational context of an applications’ usage.

**Existing solution for Problem 2**

In regards to the “premature” time-out, a client need to use the refresh token functionality of OpenID Connect.

When logging in, a user gets an Access token, that can be used for accessing services and resources. This Access token only lives for 5 minutes (and this will never change to more minutes). This token can be used to get a new Access token from Keycloak (without logging in again) before the client send it to the services. The Refresh token currently lives for 30 minutes and the timer will reset when the token is used. So the user should just “do something” in the app every 30 minutes to not get logged out in this solution.

The refreshing token functionality for Java and x-javascript, Keycloak have documentation here:

https://www.keycloak.org/docs/3.3/securing\_apps/topics/oidc/oidc-overview.html

In x-javascript used in the Portal, it is as simple as:

|  |
| --- |
| keycloak.updateToken(30).success(function() {      loadData();  }).error(function() {      log('Failed to refresh token');      doLogout();  ); |