**CAS Authentication configuration for USM Back-end**

# Introduction

This document presents:

* the configuration needed to transform USM Back-end to a CAS-enabled application in order to communicate/authenticate using CAS server
* the configuration needed by application servers where USM application will be deployed
* how to quick test that USM Back-end is CAS protected application

# Assumptions

CAS (<http://jasig.github.io/cas/>) server is already installed, configured and available within the premises where the USM back-end application is about to be deployed.

# References

1. **CAS Documentation**: <http://jasig.github.io/cas>

# Transform USM Back-end to a CAS-enabled application

Performing this transformation can be achieved by adding CAS profile when building the application for the desired environment. Enabling this profile will transform USM based on the following properties specified in /UNIONVMS/trunk/USM/java/administration/rest/pom.xml file:

1. CAS.casServerUrl - CAS server URL
2. CAS.serverName – application server host and port where USM Back-end will be deployed. This property can be set by following implicit examples provided in pom file depending on target application server:
   1. WebLogic: <CAS.serverName>http://${wls.host}:${wls.port}</CAS.serverName>
   2. JBoss(WildFly): <CAS.serverName>http://${wf.host}:${wf.port}</CAS.serverName>

The following commands can be used to enable CAS on USM back-end:

1. Weblogic: /dev/mare/unionvms/trunk/USM/java/administration $ mvn clean install –PCAS,swagger,weblogic –DskipTests
2. JBoss(WildFly): /dev/mare/unionvms/trunk/USM/java/administration $ mvn clean install –PCAS,swagger,wildfly –DskipTests

Transforming USM Back-end to a CAS-enabled application consists in two modifications:

1. Adding new filters to web.xml deployment descriptor

<!-- The following security constraint enables CAS ${enable.CAS.end}

<filter>

<filter-name>CAS Authentication Filter</filter-name>

<filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>

<init-param>

<param-name>casServerLoginUrl</param-name>

<param-value>${CAS.casServerUrl}/login</param-value>

</init-param>

<init-param>

<param-name>serverName</param-name>

<param-value>${CAS.serverName}</param-value>

</init-param>

<init-param>

<param-name>authenticationRedirectStrategyClass</param-name>

<param-value>eu.europa.ec.mare.usm.administration.cas.CASAuthenticationRedirectStrategy</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CAS Authentication Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter>

<filter-name>CAS Validation Filter</filter-name>

<filter-class>org.jasig.cas.client.validation.Cas10TicketValidationFilter</filter-class>

<init-param>

<param-name>casServerUrlPrefix</param-name>

<param-value>${CAS.casServerUrl}</param-value>

</init-param>

<init-param>

<param-name>serverName</param-name>

<param-value>${CAS.serverName}</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>CAS Validation Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<filter>

<filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

<filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

${enable.CAS.start} End of CAS security constraint-->

1. Adding a new custom CASAuthenticationRedirectStrategy (used by CAS authentication filter to redirect when authentication is required). In order to maintain code clean and with minimum dependencies this class is compiled/used only when CAS profile is enabled. This class will extend default redirect strategy of CAS: whenever the request is not authenticated and it is done using X-Requested-With:XMLHttpRequest header the response will be a JSON object { "success" : false, "status" : "ECAS\_AUTHENTICATION\_REQUIRED", "code" : 303, "message" : "session expired" }:

@Override

**public** **void** redirect(HttpServletRequest request, HttpServletResponse response, String potentialRedirectUrl) **throws** IOException {

**if** ("XMLHttpRequest".equalsIgnoreCase(request.getHeader("X-Requested-With"))) {

response.setContentType("application/json");

response.setStatus(200);

**final** PrintWriter writer = response.getWriter();

writer.write("{ \"success\" : false, \"status\" : \"CAS\_AUTHENTICATION\_REQUIRED\", \"code\" : 303, \"message\" : \"session expired\" }");

} **else** {

response.sendRedirect(potentialRedirectUrl);

}

}

}

# Configure application servers to support running of CAS-enabled applications

Application servers where USM Back-end is deployed need to have in their trust stores certificates of CAS server. There are two possible solutions to achieve this:

1. Import CAS server certificate into JVM trust store used by AS
2. Configure AS to use its own trust store and import CAS server certificate in it

The following command can be used to import a certificate into a trust store (JVM or AS):

*keytool -import -alias ca -file somecert.cer -keystore cacerts -storepass changeit*

Beside this WebLogic needs extra configuration in order to be able to communicate with CAS server (use TLS1 protocol instead of TLS1.2). In order to achieve this following command should be used in Weblogic server start-up script:

1. UNIX $DOMAIN\_HOME/bin/startWebLogic.sh

# START WEBLOGIC

echo "starting weblogic with Java version:"

${JAVA\_HOME}/bin/java ${JAVA\_VM} -version

export ECAS\_OPTIONS="-Dweblogic.wsee.client.ssl.usejdk=true -DUseSunHttpHandler=true -Dhttps.protocols=TLSv1 -Dweblogic.security.SSL.enableJSSE=true -Dweblogic.ssl.JSSEEnabled=true -Dweblogic.security.SSL.protocolVersion=TLS1"

export JAVA\_OPTIONS=$JAVA\_OPTIONS:$ECAS\_OPTIONS

1. Windows %DOMAIN\_HOME%/bin/startWebLogic.cmd

@REM START WEBLOGIC

echo starting weblogic with Java version:

%JAVA\_HOME%\bin\java %JAVA\_VM% -version

SET ECAS\_OPTIONS=-Dweblogic.wsee.client.ssl.usejdk=true -DUseSunHttpHandler=true -Dhttps.protocols=TLSv1 -Dweblogic.security.SSL.enableJSSE=true -Dweblogic.ssl.JSSEEnabled=true -Dweblogic.security.SSL.protocolVersion=TLS1

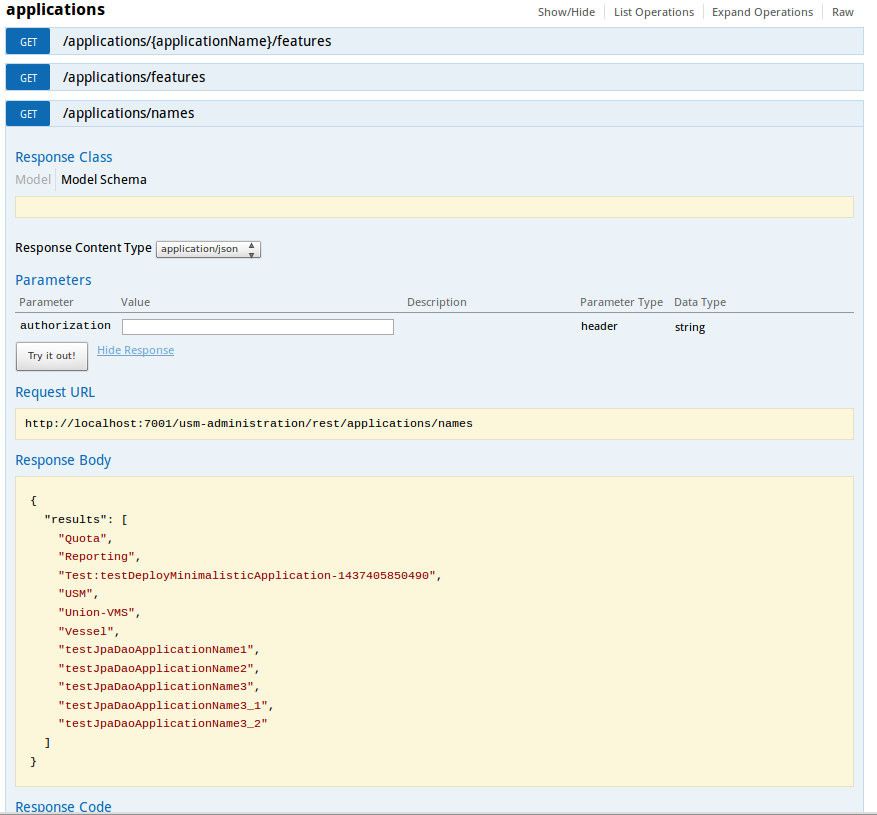
SET JAVA\_OPTIONS=%JAVA\_OPTIONS% %ECAS\_OPTIONS%

# Testing CAS-enabled USM Back-end application

After deploying CAS-enabled USM back-end application with swagger profile enabled access the following link:

1. JBoss (WildFly): http:/localhost:8080/usm-administration/
2. WebLogic: localhost:7001/usm-administration/

When the link is accessed for the first time there should be a redirect to CAS login page. Once the login operation is successful there should be a return link on CAS server to usm-administration page and now the swagger APIDOCS page should be available. Browse to one of the GET operation listed there and try to use it without providing the JWT token in the **authorisation** header field (see below screenshot)

Another test will be to take the request URL from swagger page and enter it into Google Advanced Rest Client application. Invoking the GET operation should return the CAS login page (with a redirect in the response trace). By adding the X-Requested-With:XMLHttpRequest header the response should change to JSON object { "success" : false, "status" : "ECAS\_AUTHENTICATION\_REQUIRED", "code" : 303, "message" : "session expired" }