USM setup

Contents

[Prerequisite 3](#_Toc431397945)

[Standalone 3](#_Toc431397946)

[Wildfly - PostgreSQL 3](#_Toc431397947)

[Setup the PostgreSQL server 3](#_Toc431397948)

[Install and configure Wildfly 3](#_Toc431397949)

[Create USM2 data source 4](#_Toc431397950)

[Build and deploy the application 5](#_Toc431397951)

[Enable HTTPS 6](#_Toc431397952)

[Weblogic - Oracle 7](#_Toc431397953)

[Setup the Oracle server 7](#_Toc431397954)

[Install and configure Weblogic 12.1.2 7](#_Toc431397955)

[Create USM2 data source 7](#_Toc431397956)

[Build and deploy the application 8](#_Toc431397957)

[Enable HTTPS 8](#_Toc431397958)

[CAS Authentication configuration for USM Back-end 9](#_Toc431397959)

[Introduction 9](#_Toc431397960)

[Assumptions 10](#_Toc431397961)

[References 10](#_Toc431397962)

[Configure the USM Back-end for CAS authentication 10](#_Toc431397963)

[Testing CAS-enabled USM Back-end application 14](#_Toc431397964)

[ECAS Authentication Provider Setup for USM Back-end 15](#_Toc431397965)

[Introduction 15](#_Toc431397966)

[Assumptions 15](#_Toc431397967)

[References 15](#_Toc431397968)

[ECAS Authentication Provider Installation Procedure 15](#_Toc431397969)

[WebLogic domain configuration 16](#_Toc431397970)

[Configure the USM Back-end for ECAS authentication 17](#_Toc431397971)

[Configuring the USM Back-end for ECAS authentication is described in general terms in section 6. *PROTECT YOUR WEB APPLICATION* of the ECAS Client *Installation and Configuration Guide – Basic* (see References). In a nutshell, performing this configuration entails at the minimum the execution of the following three actions: 17](#_Toc431397972)

[Testing CAS-enabled USM Back-end application 19](#_Toc431397973)

# Prerequisite

1. Check out the USM source code from the SVN server (for example USM\_SRC).

In order to do this, a SVN client is needed to be installed.

1. Download and install JDK 7 and Apache Maven 3.3.3
2. Set the JAVA\_HOME and the M2\_HOME system variables.

Set the PATH, i.e. add to the system path %JAVA\_HOME%\bin;%M2\_HOME%\bin;

Also, the Nexus repository must be added to the settings.xml file (attachment settings.xml)

# Standalone

## Wildfly - PostgreSQL

### Setup the PostgreSQL server

1. Install the PostgreSQL server 9.1.
2. An **usm2** role with password **password** with access to an **usm2** database must be created.
3. Go to USM\_SRC/database/liquibase and execute

**mvn liquibase:update -Ppostgres,local,testdata**

### Install and configure Wildfly

1. Get the application server distribution for 8.2.0 from <http://wildfly.org/downloads/>
2. Uncompress and open readme.txt for starting instructions (run <JBOSS\_HOME>\bin\standalone.bat on Windows)
3. Access the web console at <http://localhost:9990/console> -> you should get a page telling you that you have to configure users
4. Follow these instructions to create a management user (This is user must be added in settings.xml in the wf-local profile)

wildfly-8.2.0.Final>bin\add-user.bat

What type of user do you wish to add?

a) Management User (mgmt-users.properties)

b) Application User (application-users.properties)

(a): a

Enter the details of the new user to add.

Using realm 'ManagementRealm' as discovered from the existing property files.

Username : wildfly

Password recommendations are listed below. To modify these restrictions edit the add-user.properties configuration file.

- The password should not be one of the following restricted values {root, admin, administrator}

- The password should contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), 1 non-alphanumeric symbol(s)

- The password should be different from the username

Password :

JBAS015267: Password must have at least 1 non-alphanumeric symbol.

Are you sure you want to use the password entered yes/no? yes

Re-enter Password :

What groups do you want this user to belong to? (Please enter a comma separated list, or leave blank for none)[

]:

About to add user 'wildfly' for realm 'ManagementRealm'

Is this correct yes/no? yes

Added user 'wildfly' to file '......\wildfly-8.2.0.Final\standalone\configuration\mgmt-users.properties'

Added user 'wildfly' to file '......\wildfly-8.2.0.Final\domain\configuration\mgmt-users.properties'

Added user 'wildfly' with groups to file '......\wildfly-8.2.0.Final\standalone\configuration\mgmt-groups.properties'

Added user 'wildfly' with groups to file '......\wildfly-8.2.0.Final\domain\configuration\mgmt-groups.properties'

Is this new user going to be used for one AS process to connect to another AS process?

e.g. for a slave host controller connecting to the master or for a Remoting connection for server to server EJB calls.

yes/no? no

Press any key to continue . . .

1. Retry connecting to the console  <http://localhost:9990/console> and enter the user and password you chose
2. Download jdbc driver for postgres from <https://jdbc.postgresql.org/download.html> ([JDBC41 Postgresql Driver, Version 9.4-1201](https://jdbc.postgresql.org/download/postgresql-9.4-1201.jdbc41.jar))
3. Deploy and enable it using the web console

* Go to **deployments**
* Click Add
* Select the driver jar file and click next
* Tick the enabled box
* Click save

### Create USM2 data source

Stop the server, add the following to the datasources element to the <JBOSS\_HOME>\standalone\configuration\standalone.xml config file and restart it:

<datasource jta="true" jndi-name="java:/jdbc/USM2" pool-name="jdbc/USM2" enabled="true" use-ccm="true">

    <connection-url>jdbc:postgresql://cygnus-dev.athens.intrasoft-intl.private:5432/usm2</connection-url>

    <driver-class>org.postgresql.Driver</driver-class>

    <driver>postgresql-9.4-1201.jdbc41.jar</driver>

    <security>

        <user-name>usm2</user-name>

        <password>password</password>

    </security>

    <validation>

        <validate-on-match>false</validate-on-match>

        <background-validation>false</background-validation>

    </validation>

    <timeout>

        <set-tx-query-timeout>false</set-tx-query-timeout>

        <blocking-timeout-millis>0</blocking-timeout-millis>

        <idle-timeout-minutes>0</idle-timeout-minutes>

        <query-timeout>0</query-timeout>

        <use-try-lock>0</use-try-lock>

        <allocation-retry>0</allocation-retry>

        <allocation-retry-wait-millis>0</allocation-retry-wait-millis>

    </timeout>

    <statement>

        <share-prepared-statements>false</share-prepared-statements>

    </statement>

</datasource>

### Build and deploy the application

1. Build and deploy the application back end

Go to USM\_SRC /java and execute (the application server must be started)

**mvn clean install –DskipTests**

1. Build and deploy the application front end

If your environment is behind a proxy edit .bowerrc file located in \administration\src\main\webapp folder to set proxy variables.

If you are not behind a proxy remove these variables.

{

"directory" : "bower\_components",

"proxy":"http://proxy:port",

"https-proxy":"http://proxy:port"

}

Go to USM\_ SRC /web/administration and execute

**mvn clean package**

1. Deploy and enable the USM\_SRC /web/administration/target/administration.war using the web console

* Go to **deployments**
* Click Add
* Select the administration.war file and click next
* Tick the enabled box
* Click save

### Enable HTTPS

In order for your environment to work with https you need to modify your proxies section of the gruntfile.js or the respective proxies.yaml file.

Add or modify the values of the following parameters:

port: 8443 (this is the secure port for the wildfly)

https: true

secure: false (this is to avoid self-signed certificate from being verified)

**Enable HTTPS on WildFly**

1. Edit file [WILDFLY\_HOME]/standalone/configuration/standalone.xml and

* add the following into the management/securiy-realms

<security-realm name="SSLRealm">

<server-identities>

<ssl>

<keystore path="localhost.jks" relative-to="jboss.server.config.dir" keystore-password="changeit" alias="localhost" key-password="changeit"/>

</ssl>

</server-identities>

</security-realm>

* add the following into <subsystem xmlns="urn:jboss:domain:undertow:1.2">/<server name="default-server">

<https-listener name="https" socket-binding="https" security-realm="SSLRealm"/>

1. Start WildFly
2. Access <https://localhost:8443/>

To access the REST service on the aforementioned instance from java code:

1. Save the attached localhost.jks in your [WILDFLY\_HOME]/standalone/configuration/ directory
2. Save the localhost.cer to a directory of your choice (e.g. C:\temp)

$JAVA\_HOME\jre\bin\keytool -export -v -alias localhost -file C:\temp\localhost.cer -keystore [WILDFLY\_HOME] \standalone\configuration\localhost.jks -storepass changeit

1. Import the certificate into you preferred JDK trusted certificate list:

$JAVA\_HOME\jre\bin\keytool -import -trustcacerts -keystore $JAVA\_HOME\jre\lib\security\cacerts -storepass changeit -noprompt -alias localhost -file C:\temp \localhost.cer

1. Restart the server

## Weblogic - Oracle

### Setup the Oracle server

1. Install the Oracle server 11g
2. An **usm2** user with password **password** with access to an **usm2** database must be created.
3. Go to USM\_SRC/database/liquibase and execute

**mvn liquibase:update -Poracle,local,testdata**

### Install and configure Weblogic 12.1.2

1. Get the application server distribution for 12.1.2 and install it.
2. Execute ${WL\_HOME}\oracle\_common\common\bin\config.bat

A new window will be displayed. Select “Create a new domain” and click **Next**.

Fill in the **Name** and the **Password** for the **Administrator Account** (for example weblogic/weblogic1) and click **Next** (This user must be added in settings.xml in the wls-local profile).

Finish the installation depending to your needs.

1. Start the Weblogic server by hitting ${WL\_HOME}\user\_projects\domains\base\_domain\startWebLogic
2. Start the node manager by hitting ${WL\_HOME}\user\_projects\domains\base\_domain\bin\startNodeManager
3. Access the web console at <http://localhost:7001/console/> and log-in using the username and the password that you just add during the installation.
4. Enable tunneling on Weblogic

Click **Servers** link (left panel), click **AdminServer** link.

Go to **Configuration/General** tab, fill in the **Listen Address** filed(if it is the case).

Go to **Protocols/General tab**, and check **Enable Tunneling**.

Click **Save**.

### Create USM2 data source

1. Click **Services/Data Sources** link (left panel).
2. Click **New/Generic Data Source**. Fill in a name and add **jdbc/USM2** as JNDI Name. Click **Next**.
3. Choose the **Database Driver** and click **Next.** Click **Next** again.
4. Fill in the **Database Name, Host Name, Port, Database User Name, Password, Confirm Password** fields and click **Next**.
5. In the next screen click **Test Configuration**. The Weblogic console will notify you if the connection was successful. Click **Next** and in the next screen choose **AdminServer** in the **Servers** section. This will link the data source to the server.
6. Click **Finish**.

### Build and deploy the application

1. Build and deploy the application back end

Go to USM\_src /java and execute (the application server must be started)

**mvn clean install -Pweblogic,swagger -DskipTests**

1. Build and deploy the application front end

If your environment is behind a proxy edit .bowerrc file located in \administration\src\main\webapp folder to set proxy variables.

If you are not behind a proxy remove these variables.

{

"directory" : "bower\_components",

"proxy":"http://proxy:port",

"https-proxy":"http://proxy:port"

}

Go to USM\_SRC /web/administration and execute

**mvn clean package -Pweblogic –DskipTests**

Deploy and enable the USM\_SRC /web/administration/target/administration.war using the web console

* + Click **Deployments** link (left panel) .
  + Click **Install**
  + Select the USM\_src \web\administration\target \administration.war file and click **Next**
  + Finish the installation by clicking **Next** and **Finish** and the end.

### Enable HTTPS

In order for your environment to work with https you need to modify your proxies section of the gruntfile.js or the respective proxies.yaml file.

Add or modify the values of the following parameters:

port: 8443 (this is the secure port for the wildfly)

https: true

secure: false (this is to avoid self-signed certificate from being verified)

**Enable HTTPS on Weblogic**

1. Click **Servers** link (left panel), click **AdminServer** link.

Go to **Configuration/General** tab and check **SSL Listen Port Enabled.**

Click **Save**.

To access the REST service on the aforementioned instance from java code:

1. Save the attached localhost.jks in your ${WL\_HOME}\user\_projects\domains\base\_domain\ directory
2. Save the localhost.cer to a directory of your choice (e.g. C:\temp)

$JAVA\_HOME\jre\bin\keytool -export -v -alias localhost -file C:\temp\localhost.cer -keystore ${WL\_HOME}\user\_projects\domains\base\_domain\localhost.jks -storepass changeit

1. Import the certificate into you preferred JDK trusted certificate list:

$JAVA\_HOME\jre\bin\keytool -import -trustcacerts -keystore $JAVA\_HOME\jre\lib\security\cacerts -storepass changeit -noprompt -alias localhost -file C:\temp \localhost.cer

1. Restart the server

# CAS Authentication configuration for USM Back-end

## Introduction

This chapter presents:

1. the configuration needed to transform USM Back-end to a CAS-enabled application in order to communicate/authenticate using CAS server
2. the configuration needed by application servers where USM application will be deployed
3. 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>

## Configure the USM Back-end for CAS authentication

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:

* WebLogic: <CAS.serverName>http://${wls.host}:${wls.port}</CAS.serverName>
* 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: USM\_SRC/java/administration

**$ mvn clean install -PCAS,swagger,weblogic -DskipTests**

1. JBoss(WildFly): USM\_SRC/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](http://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);

                }

        }

        }

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):

$JAVA\_HOME\jre\bin\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/](http://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.

# ECAS Authentication Provider Setup for USM Back-end

## Introduction

This chapter presents:

1. how to install the ECAS Authentication Provider
2. how to configure the USM Back-end for ECAS authentication on the basis that:
   * the information provided may be used to restrict access to the USM front-end (more suitable for ECAS authentication since it is targeted to human beings) to end-users authenticated via ECAS.
   * the information provided may be used to restrict access to both the USM front-end and back-end in the case where both components (front and back) are packaged in a single WAR (using the back-end WAR as an overlay for the front-end) or in a single EAR (presumably with shared session configuration)

## Assumptions

As ECAS is an European Commission product that only exists inside the European Commission. It is assumed that the ECAS Authentication Provider and the USM Back-end are to be deployed onto an Oracle Weblogic Application Server instance (primary J2EE platform used by the European Commission).

## References

1. **ECAS Documentation**:

<https://webgate.ec.europa.eu/CITnet/confluence/display/IAM/ECAS>

1. **ECAS Repository**: <https://webgate.ec.europa.eu/CITnet/confluence/display/IAM/ECAS+Forge>
2. **ECAS Client Installation and Configuration Guide – Basic***:*

[Installation and Configuration Guide - Basic.pdf](https://webgate.ec.europa.eu/CITnet/svn/ecas-public/clients/java/tags/4.3.1/doc/ECAS%20Client%20Installation%20and%20Configuration%20Guide%20-%20Basic.pdf)

## ECAS Authentication Provider Installation Procedure

1. Make a backup copy of your Weblogic domain configuration (config.xml and start-sup shell scripts)
2. Download the latest/desired ECAS Client version (currently 4.3.1) from the ECAS Repository (see *References*). This includes:
   * **ecas-weblogic-10.3-authprovider-4.3.1.jar**: The ECAS Client JAR EcasIdentityAsserterV2 for WebLogic Server 10.3 and above
   * **ecas-demo.ear**: The demo application (optional)
   * **security.properties**: A customized resource bundle to provide descriptions for the EcasIdentityAsserterV2 provider specific page in the WebLogic admin console.
   * **log4j-1.2.15.jar**: a patched log4j JAR
   * **log4j.xml**: The example log4j configuration file (to be adapted to your environment)
3. Download the ECAS Client *Installation and Configuration Guide – Basic* from the ECAS Repository (see *References*).
4. Install the ECAS Client as described in the *Installation and Configuration Guide – Basic* document. In particular, do make sure that you follow the instructions provided in Section 4.3. *Security requirements* of the aforementioned document to:
   * **Enable strong cryptography**
   * **Disable SSLv3** (see Sample WebLogic start-up script)
   * **Disable SSLv3 for WebLogic-generated Web Service clients** (necessary on Weblogic 12.1.1 but not on version 12.1.2)
5. After completing the installation of the ECAS Client, make sure you take a note of the fact that the Weblogic administration console is now accessible only via the <http://localhost:7001/console/login/LoginForm.jsp> URL (unless your weblogic administrator user is defined in ECAS user database).
6. If your Weblogic server/domain is deployed outside the European Commission, you will most probably integrate with an ECAS Mockup Server instance rather than with the (real) ECAS server instance with which the ECAS Client is (pre-)configured to integrate. If this is the case, you should open the WebLogic administration console (see above), navigate to the ***Security Realms >myrealm >Providers >EcasIdentityAsserterV2*** screen and update the following properties in the ***Provider Specific*** panel (see **Sample WebLogic configuration**:

* **Assurance Level**: LOW (unless you need to accept internal EC users only)
* **Accept Strengths**: BASIC
* **Ecas Base Url**: base URL of your ECAS Server Mockup instance (e.g. <https://svm-midway.athens.intrasoft-intl.private:7012>)
* **Ecas Server Direct Host Name**: host name (or IP address) of your ECAS Server Mockup instance (e.g. *svm-midway.athens.intrasoft-intl.private*)
* **Ecas Server Direct One Way Ssl Port**: port number of your ECAS Server Mockup instance (e.g. *7012*)

1. Restart your Weblogic server instance for the latest changes to take effect.

## WebLogic domain configuration

1. **Sample WebLogic configuration**

The below example displays the suggested configuration for EcasIdentityAsserterV2 in the Weblogic domain configuration file ($DOMAIN\_HOME/config/config.xml).

<sec:authentication-provider xmlns:sch="https://www.cc.cec/cas/schemas" xsi:type="sch:**ecas-identity-asserter-v2Type**">   
<n1:name xmlns:n1="http://www.bea.com/ns/weblogic/90/security">EcasIdentityAsserterV2</n1:name>   
<sch:control-flag>**OPTIONAL**</sch:control-flag>   
<sch:ecas-base-url>[**https://svm-midway.athens.intrasoft-intl.private:7012**](https://svm-midway.athens.intrasoft-intl.private:7012)</sch:ecas-base-url>   
<sch:accept-strengths>**BASIC**</sch:accept-strengths>   
<sch:assurance-level>**LOW**</sch:assurance-level>   
<sch:ecas-server-direct-host-name>**svm-midway.athens.intrasoft-intl.private**</sch:ecas-server-direct-host-name>   
<sch:ecas-server-direct-one-way-ssl-port>**7012**</sch:ecas-server-direct-one-way-ssl-port>   
</sec:authentication-provider>

The below example highlights the mandatory configuration for DefaultAuthenticator in the Weblogic domain configuration file ($DOMAIN\_HOME/config/config.xml).

<sec:authentication-provider xsi:type="wls:**default-authenticatorType**">   
<sec:control-flag>**OPTIONAL**</sec:control-flag>   
<wls:minimum-password-length>5</wls:minimum-password-length>   
</sec:authentication-provider>

1. **Sample WebLogic start-up script**

The below example highlights the mandatory configuration to disable SSLv3, defined in the Weblogic server start-up script ($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**   
  
if [ "${WLS\_REDIRECT\_LOG}" = "" ] ; then   
echo "Starting WLS with line:"   
echo "${JAVA\_HOME}/bin/java ${JAVA\_VM} ${MEM\_ARGS} -Dweblogic.Name=${SERVER\_NAME} -Djava.security.policy=${WL\_HOME}/server/lib/weblogic.policy ${JAVA\_OPTIONS} ${PROXY\_SETTINGS} ${SERVER\_CLASS}"   
${JAVA\_HOME}/bin/java ${JAVA\_VM} ${MEM\_ARGS} -Dweblogic.Name=${SERVER\_NAME} -Djava.security.policy=${WL\_HOME}/server/lib/weblogic.policy ${JAVA\_OPTIONS} ${PROXY\_SETTINGS} ${SERVER\_CLASS}   
else   
echo "Redirecting output from WLS window to ${WLS\_REDIRECT\_LOG}"   
${JAVA\_HOME}/bin/java ${JAVA\_VM} ${MEM\_ARGS} -Dweblogic.Name=${SERVER\_NAME} -Djava.security.policy=${WL\_HOME}/server/lib/weblogic.policy ${JAVA\_OPTIONS} ${PROXY\_SETTINGS} ${SERVER\_CLASS} >"${WLS\_REDIRECT\_LOG}" 2>&1   
fi

## Configure the USM Back-end for ECAS authentication

## Configuring the USM Back-end for ECAS authentication is described in general terms in section 6. PROTECT YOUR WEB APPLICATION of the ECAS Client Installation and Configuration Guide – Basic (see References). In a nutshell, performing this configuration entails at the minimum the execution of the following three actions:

1. Add a security constraint to the **web.xml** deployment descriptor defined in the Administration-Rest-Service as depicted below

<?xml version="1.0" encoding="UTF-8"?>   
  
<web-app xmlns="http://java.sun.com/xml/ns/javaee"   
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"   
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee <http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd>"   
version="3.0">   
<session-config>   
<session-timeout>30</session-timeout>   
</session-config>   
  
**<security-constraint>**   
**<web-resource-collection>**   
    **<web-resource-name>protected</web-resource-name>**   
      <description> Protected REST services</description>   
      **<url-pattern>\*</url-pattern>**   
    **</web-resource-collection>**   
    **<auth-constraint>**   
      <description>   
        Requires users to be authenticated but   
        does not require them to be authorized.   
      </description>   
      **<role-name>**\***</role-name>**   
    **</auth-constraint>**   
    <user-data-constraint>   
      <description>Encryption is not required for this area.</description>   
      <transport-guarantee>NONE</transport-guarantee>   
    </user-data-constraint>   
  **</security-constraint>**   
</web-app>

1. Add an allow-all-roles option to the **weblogic.xml** deployment descriptor defined in the Administration-Rest-Service as depicted below

<?xml version="1.0" encoding="UTF-8"?>   
<weblogic-web-app xmlns="http://xmlns.oracle.com/weblogic/weblogic-web-app">   
  
  <context-root>usm-administration</context-root>   
  <container-descriptor>   
    **<!-- Allow some resources to be accessed by**   
    **authenticated users who do not possess any role -->**   
    **<allow-all-roles>true</allow-all-roles>**   
    <prefer-web-inf-classes>true</prefer-web-inf-classes>   
  </container-descriptor>   
</weblogic-web-app>

1. Build and deploy the Administration-Rest-Service without executing the integrations tests as depicted below

**mvn clean install -Pswagger,weblogic –DskipTests**

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

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

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

$JAVA\_HOME\jre\bin\keytool -import -alias ca -file somecert.cer -keystore cacerts -storepass changeit

## Testing CAS-enabled USM Back-end application

At this point you should be able to the USM back-end RESTful web-services without providing any JWT token in the **authorisation** header field (customarily used for user authentication), simply by entering the service URL (e.g. <http://localhost:7001/usm-administration/rest/applications/names>) directly in your browser address bar

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

1. WebLogic: <http://localhost:7001/usm-administration/rest/applications/names>

When the link is accessed for the first time there should be a redirect to ECAS login page. Once the login operation is successful there should be a return link on ECAS server to usm-administration page.