Support SAML v2

Ephesoft v4.1.1.0 enables you to configure Ephesoft with Spring Security, both with and without SAML (Security Assertion Markup Language) integration.

SAML is an XML-based, open-standard data format for exchanging authentication and authorization data between parties, in particular, between an identity provider (IdP) and a service provider (end user).

Spring Security Framework has the following components:

- **applicationContext-security.xml**: User can configure all Spring Security related beans from this xml.
- **SAML IdP**: A third party IdP (Identity Provider) is required to configure the AuthenticationHandler. Metadata from Ephesoft is required to be registered with the IdP.
- **JKS Keystore**: A Keystore needs to be set up for general authentication.
- **Logout URL**: This parameter is required to redirect the user to a page when they click on the Ephesoft logout button.
- **Mode**: This parameter can have two values; **AUTHENTICATION_ONLY** and **AUTHENTICATION_AUTHORIZATION**. Based on this parameter, Ephesoft Authorization is used. If the **AUTHENTICATION_ONLY** parameter is used, then provided username is used for authorization using configured connectivity. If **AUTHENTICATION_AUTHORIZATION** is used, Ephesoft looks for Roles and IsSuperAdmin attributes in the SAML Response. The attribute names that are passed through SAML response are configurable.

Configurations for Integrating SAML Enabled Spring Security Framework in Ephesoft

The following files need to be configured:

- EPHESOFT_HOME/Application/applicationContext.xml.
- EPHESOFT_HOME/Application/WEB-INF/web.xml.
- EPHESOFT_HOME/application/WEB-INF/classes/security/samlKeystore.jks
- EPHESOFT_HOME/Application/WEB-INF/classes/META-INF/applicationContext-security.xml

applicationContext.xml

Uncomment **`applicationContext-security.xml`** bean in **applicationContext.xml** to use SAML SSO.

```
applicationContext.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:p="http://www.springframework.org/schema/p"
    xmlns:util="http://www.springframework.org/schema/util" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop" xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                        http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd
                        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd
                        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
                        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
                            "
    default-autowire="byName">

    <!-- Uncomment to Use SAML SSO -->
    <import resource="classpath:/META-INF/applicationContext-security.xml" />

    <!-- Spring file 'applicationContext-workflows.xml' is used to run the Ephesoft application.          -->
    <!-- This line should not be removed or commented.                                                    -->
    <import resource="classpath:/META-INF/applicationContext-twain-scanner.xml" />
```

web.xml

- Uncomment the **springSecurityFilterChain** filter and its filter mapping.
- Comment out the **sessionTimeoutFilter** and its filter mapping
- Comment out the **SessionTimeoutServlet** and its Servlet Mapping
- Comment out all **security-constraints** and **login-config** nodes

Logout url needs to be of the form: **http://{hostname}:{port-number}/dcma/saml/logout**

```
        <!-- Logout URL -->
        <init-param>
            <param-name>logoutUrl</param-name>
            <param-value>http://ephesoftvm:8080/dcma/saml/logout</param-value>
        </init-param>
    </filter>
```

- Select appropriate value for **authenticationType** as per below logic:
    - If only username will be passed along as attribute in the SAML message and authorization will be determined by Ephesoft.
    - If username, group and isSuperAdmin values will be passed along as attributes in the SAML message.

```
    <!-- Parameter added for choosing authentication type from multiple authentication type supported by Ephesoft
        0:Ephesoft Authentication, 1:SSO authentication only, 2:SSO authentication and authorisation with session variables -->
    <context-param>
        <param-name>authenticationType</param-name>
        <param-value>1</param-value>
    </context-param>
```

SamlKeystore.jks

A Java Key Store (JKS) is a repository of security certificates. SAML exchanges involve usage of cryptography for signing and encryption of data. This Key store contains **certificate and private key** that are used for digitally signing the SAML messages and encrypt their content.

A Default Keystore is provided in application. However, user can create custom Keystores as well.

Creating a Private Key

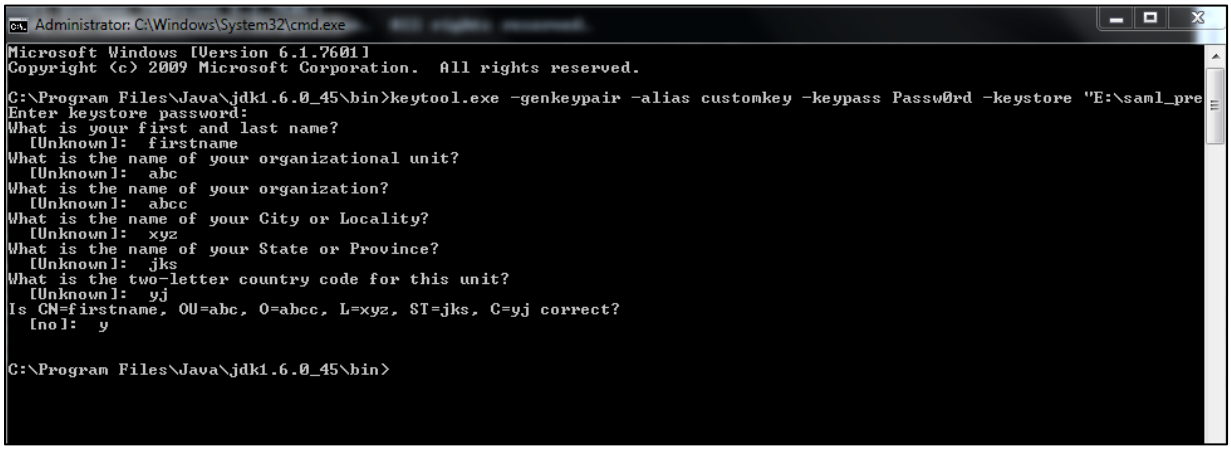
1. Open command prompt at key tool path and run the following command for generating private key for samlKeyStore.

   C:\Program Files\Java\jdk1.7.XXXX\bin>keytool.exe -genkeypair -alias "alias for private key" -keypass "password for key" -keystore "path to keystore"

   You are prompted for Keystore password.

2. Enter **nalle123**.
3. Complete the following prompts as applicaple to your organization.

   After conforming all the added information, private key is added to the Keystore.



4. Replace this newly created samlKeystore.jks file with already placed samlKeystore.jks file at location:

   EPHESOFT_HOME/application/WEB-INF/classes/security/samlKeystore.jks.

It is recommended that you note down the alias for the private key created and corresponding password. It will be needed afterwards.

applicationContext-security.xml

- In the bean **epheSamlFilter**, the constructor args need to be populated according to the following logic:
  - Index=0: Path of Username Attribute passed in SAML message.
  - Index=1: Path of Group Attribute passed in SAML message.
  - Index=2: Path of IsSuperAdmin Attribute passed in SAML message.
- In the beans **successRedirectHandler** and **failureRedirectHandler**, user can set urls for successful and failed authentications respectively. By default, it is configured as **/home.html**

- In the bean **`keyManager`**, the constructor args need to be populated according to the following logic:
    - **First constructor argument**: The path of samlkeystore needs to be added here. Default - **`classpath:security/samlKeystore.jks`**.
    - Second constructor argument: The password of Keystore needs to be added here. Default - **`nalle123`**.
    - **Third constructor argument**: The private key and corresponding password created in the previous section need to be added in the map constructor. Default - **`<entry key="apollo" value="nalle123" />`**.
    - **Fourth constructor argument**: The private key to be used needs to be provided here. Default - **`apollo`**.

We can add more than one private key in map constructor in third argument. Just mention which key you need to use for the current scenario in fourth argument.

- In the bean **`metadataGeneratorFilter`**, the property name **`entityId`** needs to be provided with a value pertaining to the user's metadata that will be uploaded at the IdP.
- In the bean **`metadata`**, link to IdP's Metadata.xml needs to be provided.

After doing all above configurations, start Ephesoft server.

Testing at SSO Circle

The following steps need to be followed for testing Ephesoft at SSO Circle (IDP).

These steps may vary from IDP to IDP.

1. Create account at SSO Circle and login.
2. Service Provider's metadata needs to be added at SSO Circle.

As Ephesoft does not provide any functionality for generating metadata, metadata can be generated online or sample metadata with custom configurations can be created. See the image below metadata reference.

- entityID should be the FQDN registered at SSO Circle.
- The certificate in X509Certificate tag should be the PEM Encoding of your keystore.
- All Location tags should be corresponding to server at which Ephesoft is installed.

Creating a Private Key

1. Open command prompt at key tool path and run the following command for generating Keystore.

   C:\Program Files\Java\jdk1.7.XXXX\bin >keytool.exe -genkey -alias mydomain -keyalg myAlgo -keystore "path where to create keystore"

   You are prompted for Keystore password.

2. Enter a desired password.

   You are prompted to re-enter the password.

3. Re-enter the password.
4. Complete the following prompts as applicaple to your organization.

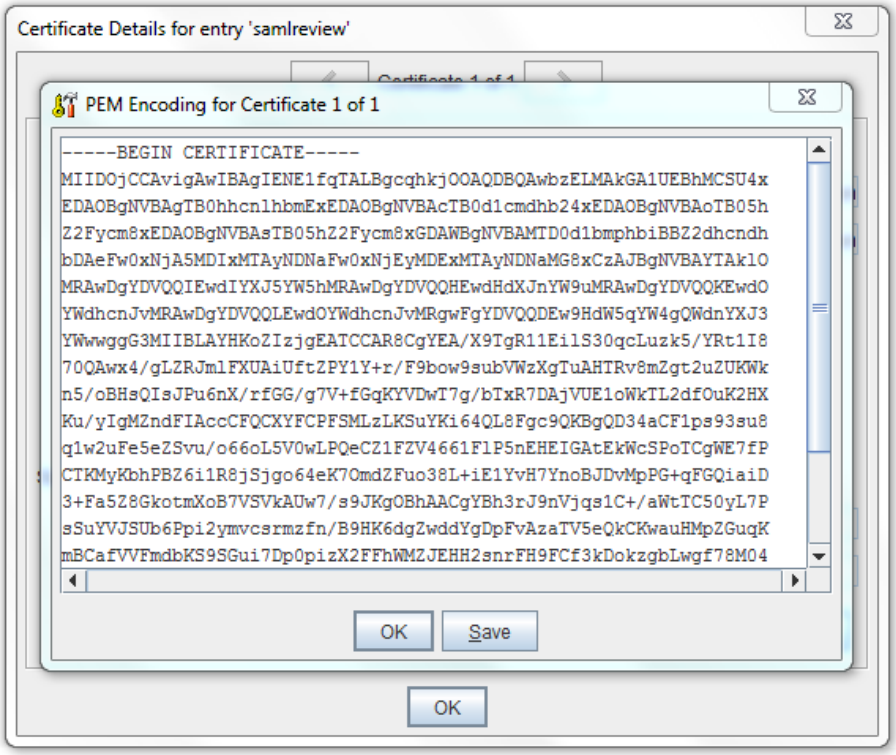
   After conforming all the added information, Keystore is created.

5. Replace this newly created Keystore file with with already placed samlKeystore.jks file at location:

   `EPHESOFT_HOME/application/WEB-INF/classes/security/samlKeystore.jks`

Getting PEM Encoding

1. Copy PEM Encoding of your newly created keystore by using online tools available. (Portecle is one such tool).



Default Keystore provided with Ephesoft can also be used.

2. Select **FirstName** while adding metadata at SSO Circle so as to get same in SAML response. This is used for authorization in Ephesoft.

3. Configure **applicationConext-security.xml** as described in the earlier section.

SSO Circle metadata is available at http://idp.ssocircle.com/idp-meta.xml.

4. Configure **web.xml** as described in the earlier section.
5. Configure **applicationConext.xml** as described in the earlier section.
6. Add the user information (First Name at SSO circle in user tag, Password at SSO Circle in password tag, and role as needed) in **tomacat-users.xml**.
7. Start Ephesoft server.

Troubleshooting

In case of any issues faced during configuration or authentication, please check the logs located at **`<Ephesoft-Installation-Directory>/Application/logs/dcma-all.log`**. Follow the instructions defined inside the logs. If the problem still persists, share the following information with engineering:

- INFO level logs
- SAZ file – collect Fiddler trace while accessing your application and authenticating to Azure AD and share SAZ file of same.

Getting Fiddler Trace

A. Install Fiddler, and start capture.

1. Go to http://fiddler2.com/home (.NET2 or .NET4 is fine).
2. Once installed, go to **Tools** > **Fiddler Options** > **Https** tab.
3. Check Decrypt HTTPS traffic, and click **Yes** for following dialogs.
4. You will receive a few pop-ups that are required in order to install the Fiddler root certificate, which allows Fiddler to sit as a man-in-the-middle to your HTTPS session. Click **Yes** to continue.
5. Click **Clear Cache** on the Fiddler toolbar.
6. Click **Edit** > **Remove All Sessions**.

B. Open your favourite browser. Verify its capturing data in Fiddler.

1. Reproduce the issue in browser.
2. Stop the Fiddler trace once done.
3. Save the fiddler trace as an SAZ file.