Services web SOAP et certificats en lien avec une connexion sécurisée (SSL)

Mise en place d'un transport sécurisé (SSL) pour les services web Adélia Studio.

La configuration de la consommation d'un service web avec Adélia Studio s'appuie sur le fichier cfgWebServices.xml.
Dans un contexte d'exécution C/Windows, il est possible de renseigner le nom d'un fichier certificat au format .pem.
Dans un contexte d'exécution Java, il est possible de renseigner le nom d'un fichier (keystore) au format .jks ou également (à partir d'Adélia Studio V12 PTF04) un fichier .pem.

Note : Depuis la version d'Adélia Studio 12 PTF04, la connexion SSL est établie automatiquement en se basant sur le fichier %adeliws%/certifs/cacert.pem pour un contexte C/Windows et
sur le fichier %java_home%/lib/jre/security/cacerts pour un contexte java.
Le fichier %adeliws%/certifs/cacert.pem (C/windows) peut - si nécessaire - être mis à jour à partir du lien suivant : https://curl.haxx.se/docs/caextract.html

Si la connexion échoue, il est possible d'activer le mode mode ServerCertAuto : <Transport><SSLCertificate><ServerCertAuto>true</ServerCertAuto></SSLCertificate></Transport>
pour forcer la récupération d'un certificat depuis le serveur (utilisation dans le cas d'un certificat auto-signé).

Guide détaillé pour la création de fichiers certificats spécifiques

***Authentification serveur (SSL 1-way*)**
Le client doit s'assurer que le serveur est bien celui qu'il prétend être.
Pour cela le serveur dispose d'un certificat signé par un CA assurant son authenticité (Pour une plate-forme de tests, il est possible d'utiliser un certificat auto-signé. Les exemples qui suivent utilisent ce type de certificat).

Configuration du serveur (tomcat) / mise en place d'une connexion sécurisée

- Création d'un certificat auto-signé (self signed) + clé privée => keystore

```
"%JAVA_HOME%\bin\keytool" -genkey -alias tomcat -keyalg RSA
```

Création par défaut du keystore *.keystore* (si inexistant) dans le répertoire %userprofile% au format JKS (JavaKeyStore) (password par défaut "changeit")

- Déclaration d'un connector https dans le fichier *server.xml* (utilisant les valeurs par défaut du keystore, keystoreFile="%userprofile%/.keystore", keystorePass="changeit", keystoreType="JKS") :

```
<Connector port="8443" maxHttpHeaderSize="8192" maxThreads="150"
minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
disableUploadTimeout="true" acceptCount="20" scheme="https"
secure="true" clientAuth="false" sslProtocol="TLS"/>
```

ou

- Déclaration d'un connector https dans le fichier *server.xml* faisant référence à un keystore particulier "c:\keystore\.keystore":

```
<Connector port="8443" maxHttpHeaderSize="8192" maxThreads="150"
minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
disableUploadTimeout="true" acceptCount="20" scheme="https"
secure="true" clientAuth="false" sslProtocol="TLS"
keystoreFile="c:/keystore/.keystore" keystorePass="pwd"/>
```

- Configuration du web.xml de l'application web hébergeant le service web => toutes les ressources sont sécurisées :

```
<security-constraint>
   <web-resource-collection>
      <web-resource-name>Wildcard means whole app requires authentication</web-resource-name>
      <url-pattern>/*</url-pattern>
   </web-resource-collection>
   <user-data-constraint>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
   </user-data-constraint>
</security-constraint>
```

Rappel : Le fichier de configuration Axis2.xml se trouvant dans l'application web hôte doit déclarer un <transportSender> et un <transportReceiver> pour le protocole https.

Consommation d'un service avec la connexion sécurisée
Plate-forme C/Windows

- Récupération du certificat (auto-signé) du serveur pour valider la connexion avec le serveur (handshake SSL)

Pour que le client accepte/valide le certificat (self-signed) du serveur, le client doit passer à ce dernier ce même certificat. Il faut récupérer ce certificat dans un format PEM (texte encodé en base64) et non au format JKS (format binaire standard java).
Il est possible de récupérer ce certificat depuis le client à l'aide de la commande *openssl* suivante :

```
openssl s_client -connect serveur:port > svca.pem
```

Il faut alors éditer le fichier Cerfile.pem et conserver uniquement la partie entre les bornes -----BEGIN CERTIFICATE----- et -----END CERTIFICATE-----

Note : les bornes doivent figurer dans le fichier.

On peut alors tester la validité du certificat avec la commande suivante :

```
Openssl s_client -connect serveur:port –CAfile svca.pem
```

La commande doit se terminer par => Verify return code: 0 (ok)

- Configuration : utilisation du certificat pour invoquer le service web

Pour utiliser le certificat lors de l'invocation du service, il faut préciser le nom du fichier dans l'élément <server_cert> du fichier *cfgWebServices.xml*.

Exemple :

```
<Service>
   <AdeliaName>AX2_DOL</AdeliaName>
   <Epr>https://vmwas7:8443/axis2/services/AX2_DOL</Epr>
   <Transport>
<SSLCertificate>
        <ServerCertFile>c:\certifs\svca.pem</ServerCertFile>
      </SSLCertificate>
   </Transport>
</Service>
```

Rappel : Le fichier de configuration Axis2.xml présent sur le client (%adeliws%/axis2) doit déclarer un <transportSender> et <transportReceiver> pour le protocole https.

Plate-forme Java

Pour que le client accepte/valide le certificat (self-signed) du serveur, le client doit passer à ce dernier ce même certificat. Il faut récupérer ce certificat dans un format JKS (exporté depuis le keystore du serveur) et l'ajouter dans le keystore du poste client.

- Récupération du certificat (auto-signé) du serveur pour valider la connexion avec le serveur (handshake SSL).

```
<serveur> keytool -export -keystore c:\keystore\.keystore -alias tomcat -file tomcat.crt
```

- Si besoin créer un keystore sur le client (si absent) :

```
<client> keytool -genkey -alias salias -keyalg RSA -keystore keystore.jks
```

- Importer le certifcat dans le keystore client (avec un alias différent de celui utilisé pour stocker la clé privée) :

```
 <client> keytool -import -alias websvc -file tomcat.crt -keystore keystore.jks
```

- Configuration : utilisation du certificat pour invoquer le service web (configuration *cfgWebServices.xml)*

Exemple :

```
<Service>
   <AdeliaName>AX2_DOL</AdeliaName>
   <Epr>https://vmwas7:8443/axis2/services/AX2_DOL</Epr>
   <Transport>
         <SSLCertificate>
         <TruststoreName>c:/certifs/java/keystore.jks</TruststoreName>
         <TruststorePwd>
            <Encrypted>false</Encrypted>
            <Password>changeit</Password>
         </TruststorePwd>
      </SSLCertificate>
   </Transport>
</Service>
```

Remarques :

- Il est également possible de récupérer le fichier certificat (au format .pem) à l'aide de la commande openssl (Cf.Plate-forme C/Windows); de convertir le fichier .pem au format .der

  ```
  openssl x509 -outform der -in certificate.pem -out certificate.der
  ```

  puis de l'enregistrer dans le keystore.

  ```
  keytool -import -alias your-alias -keystore cacerts -file certificate.der
  ```

- si le pwd est crypté pour le truststore, alors la clé de cryptage est le truststorename.

Authentification serveur + client (SSL 2-ways)

Le client doit s'assurer que le serveur est bien celui qu'il prétend être.
Le serveur doit s'assurer que le client est bien celui qu'il prétend être.
En plus de la configuration présentée dans le paragraphe précédent, il faut :

Configuration du client

Plate-forme C/Windows
Créer un certificat (self-signed ou non) + clé privée.
Utilisation des éléments KEY_FILE (fichier PEM contenant le certificat + la clé privée = certificate chain file) et SSL_PASSPHRASE pour l'accès à la clé privée (= passphrase demandé lors de la création de la clé privée avec l'option *–des3*).
Création du fichier (certificate chain file) via openssl :

```
openssl req -x509 -days 365 -newkey rsa:1024 -keyout hostkey.pem -nodes -out hostcert.pem
```

La commande crée un certificat (self-signed) dans le fichier hostcert.pem et une clé privée dans hostkey.pem.
Il suffit de concaténer le fichier *hostkey.pem* au fichier *hostcert.pem* pour obtenir un nouveau fichier qui est le certificate chain file.

Remarque : la commande recherche la présence du fichier de configuration *openssl.cnf*. Pour préciser l'emplacement de ce fichier, il faut fixer la variable d'environnement OPENSSL_CONF (sans guillemet sinon cela ne fonctionne pas).
Exemple : set OPENSSL_CONF=d:\devnet\openssl-1.0.0d\bin\openssl.cnf

Plate-forme Java
Créer un certificat (self-signed ou non) + clé privée dans un keystore :

```
%JAVA_HOME%\bin\keytool -genkey -alias key1 -keyalg RSA -keystore keystore.jks
```

Exporter le certificat pour l'ajouter au truststore du serveur (tomcat) :

```
%JAVA_HOME%\bin\keytool -export -alias key1 -keystore keystore.jks -file svcli.cer
```

Configuration du serveur (tomcat)

- Au niveau du connecteur https, activer l'authentification cliente : clientAuth="**true**"

```
<Connector port="8443" maxHttpHeaderSize="8192" maxThreads="150"
minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
disableUploadTimeout="true" acceptCount="20" scheme="https"
secure="true" clientAuth="true" sslProtocol="TLS"/>
```

- Référence au *truststore* (<truststoreFile><truststorePass><truststoreType>) utilisé pour valider les certificats clients (et non au keystore qui lui est utilisé pour les certificats serveurs : le keystore et trustore utilisent par défaut le même format [JKS] et peuvent donc faire référence à un même fichier).

```
<Connector port="8443" maxHttpHeaderSize="8192" maxThreads="150"
minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
disableUploadTimeout="true" acceptCount="20" scheme="https"
secure="true" clientAuth="true" sslProtocol="TLS" keystoreFile="c:\keystore\.keystore" truststoreFile="c:
\keystore\.keystore" />
```

- Inscrire dans le truststore de tomcat le certificat client auto-signé :
  - Certificat au format PEM (créé par openssl)
    - récupérer le certificat sur le client (fichier au format PEM)
      Importer le certificat (PEM) dans le truststore de tomcat (avec conversion en JKS):

```
 keytool -import -v -trustcacerts -alias <your alias> -file <your file>.pem -keystore <your
key store>.jks -
 storepass <your storepass>
```

      L'option <trustcacerts> n'est pas obligatoire.
  - Certificat au format JKS (sans conversion)
    -

```
keytool -import -v -alias <your alias> -file <your file>.cer -keystore <your key store>.jks
-
 storepass <your storepass>
```

Exemples : nouvelle entrée dans le trustore de tomcat pour le certificat client de PS658.

```
keytool -import -v -trustcacerts -alias PS658 -file ps658.pem -keystore c:\keystore\.keystore -storepass
changeit

keytool -import -v -alias svcli -file svcli.cer -keystore c:\keystore\.keystore -storepass changeit
```

Note : Problème avec les certificats auto-signés => il faut importer dans le truststore de tomcat un certificat pour chaque client. La solution est donc de signer les certificats des clients par un même CA (le truststore tomcat incorporant alors uniquement le certificat de ce CA) : possibilité d'utiliser un CA gratuit du style de *startcom.org*.

Signer les certificats clients avec ce CA et importer le certificat racine StartCom dans le trustore tomcat => ajuster la configuration du connecteur tomcat pour faire référence au truststore (emplacement, mot de passe).

Articles connexes

- Services web SOAP et certificats en lien avec une connexion sécurisée (SSL)
- Afficher un programme via un TRAITER_PGM *POPUP_MODAL dans un cadre du programme appelant au lieu d'une nouvelle fenêtre du navigateur
- Optimisation Client-Serveur
- Services web REST - Externalisation de la configuration
- Invocation via une URL d'un service web SOAP (Axis1/Axis2) généré avec Adélia Studio