

JAFS SSL setup

by SuperBonBon

1. Server X509 Certificate generation scripts

JAFS provide scripts to generate X509 server certificate so that you can properly use SSL and TLS on your server. You'll need to execute `bin/genKeyStore.bat` under windows or `bin/genKeyStore.sh` under linux

The script will generate the following files in the `config/ssl` directory :

- `serverKeyStore.jks`: This is the keystore file that will be used by an FTP server to send the server X509 certificates to remote hosts
- `certificate.csr`: This is a certificate request that you'll need to provide to sign the server certificate by a root CA.

2. Editing the scripts to change the server certificate content

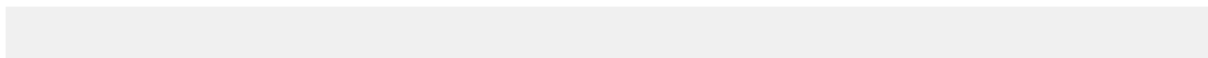
You can edit the *genKeyStore* scripts to change the certificate data, edit the following scripts with your favorite text editor.

- `CN="JAFS Administrator"` : This is the Common Name
- `OU="None"` : This is the Organization Unit
- `O="SBBI"` : This is the Organization
- `L="Somewhere"` : This is the Location
- `S="ST"` : Match the State
- `C="CT"` : The Country

Once the settings are changed simply reexecute the script to regenerate all the certificate files.

3. Changing your keystore and certificate request password

The default password of the keystore and certificate request file is 123456. If you desire to change this password you can open the certificate generation script and edit the following value :



```
set STOR_PASSWD=myNewPassword
```

You will need then to reexecute the creation script and modify the service configuration file [key-store password](#) setting:

```
<jafs-service xmlns="http://www.sbbi.net/jafs/1.0/jafs-service"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation=".....">
  ...
  <ssl-settings>
    <store-settings name="default">
      <key-store password="myNewPassword" type="jks">
        <keyStoreFile>../config/ssl/serverKeyStore.jks</keyStoreFile>
      </key-store>
      ...
    </store-settings>
  </ssl-settings>
  ...
</jafs-service>
```

4. Do I need to sign the certificate ?

No JAFS works with the self-signed generated server certificate, however if you need to provide a certification of the server identity for remote clients, you'll need to sign your certificate by a Root CA.

5. What is a Root CA

The certificate generated by the keystore generation script provided with JAFS creates a self signed certificate. Since it is self signed, it cannot be "trusted", anybody can create a certificate that contains exactly the same data.

Root Certification Authorities are organization that sign certificates, giving your certificate an authenticity. With a signed certificate you know that the data contained in the certificate can be "trusted".

To obtain a signed certificate you'll need to use the certificate.csr file with a Root CA such as thawte.com or trustcenter.de and follow the next paragraph to import your freshly new signed certificate.

6. Importing a signed certificate

JAFS SSL setup

If you decide to sign the certificate generated by the JAFS script by a root CA you'll need to integrate this certificate into a java keystore. Once you receive your signed certificate from your root CA, execute the following commands from the "bin" directory where :

- \$JAVA_HOME is the path to your JDK installation
- \$STOR_PASSWD is the keystore password (normally 123456)
- \$SIGNED_CERT_PATH the path to your signed certificate (I.E c:/certificates/myCert.pem)

```
$JAVA_HOME/bin/keytool -delete -alias JAFSServerCert -keystore
../config/ssl/serverKeyStore.jks -storepass $STOR_PASSWD
$JAVA_HOME/bin/keytool -import -trustcacerts -alias JAFSServerCert
-file $SIGNED_CERT_PATH -keystore ../config/ssl/serverKe
-storepass $STOR_PASSWD
```

You can change the JAFSServerCert alias if you desire to keep the old self-signed certificate and provide a new alias (such as SignedJAFSServerCert). If you decide to keep the old certificate do not execute the first command and change the certificate alias :

```
$JAVA_HOME/bin/keytool -import -trustcacerts -alias SignedJAFSServerCert
-file $SIGNED_CERT_PATH -keystore ../config/ssl/serverKe
-storepass $STOR_PASSWD
```

7. Managing multiple server certificates

The service configuration file allows to define multiple configurations concerning server keystore/CA keystore/CRL. Each [store-settings](#) defines a new configuration block.

You can define in each config block the CRLs that will be used to check clients certificates validity, the certificate alias in the keystore to be used by the secure connection to identify the server and also what are the allowed CA issuers for a signed client certificate.

The following example defines 2 stores configuration, the first one (myFirstStoresConfig) will:

- Use the server certificate identified by the *myCert1* alias in the ../config/ssl/serverKeyStore.jks keystore.

Note:

When no [cert-alias](#) setting is provided, the first certificate entry in the keystore will be used as the server certificate.

- Use the CA keystore ../config/ssl/serverCACerts.jks to identify signed client certificates.
- Check client certificates against 3 revocations lists (CRL1,CRL2 and CRL3), the first and second list is reloaded each 10 minutes while the third one is reloaded each 5 mins.

The second one (mySecondStoresConfig) will:

- Use the server certificate identified by the *myCert2* alias in the ../config/ssl/serverKeyStore2.jks keystore.
- Use the CA keystore ../config/ssl/serverCACerts2.jks to identify signed client certificates.

```
<ssl-settings>
  <store-settings name="myFirstStoresConfig">
    <key-store password="123456" type="jks" cert-alias="myCert1">
      <keyStoreFile>../config/ssl/serverKeyStore.jks</keyStoreFile>
    </key-store>
    <cacerts-store password="123456" type="jks">
      <caStoreFile>../config/ssl/serverCACerts.jks</caStoreFile>
    </cacerts-store>
    <crls reload-time="10">
      <crl location="http://www.bigcorp.com/CRL1.crl" name="CRL1"/>
      <crl location="http://www.bigcorp.com/CRL2.crl" name="CRL2"/>
    </crls>
    <crls reload-time="5">
      <crl location="http://www.biggercorp.com/CRL3.crl" name="CRL3"/>
    </crls>
  </store-settings>
  <store-settings name="mySecondStoresConfig">
    <key-store password="123456" type="jks" cert-alias="myCert2">
      <keyStoreFile>../config/ssl/serverKeyStore2.jks</keyStoreFile>
    </key-store>
    <cacerts-store password="123456" type="jks">
      <caStoreFile>../config/ssl/serverCACerts2.jks</caStoreFile>
    </cacerts-store>
  </store-settings>
</ssl-settings>
```

You can then assign the store settings name attribute to FTP servers using the [certsConfName](#) setting or to the HTTP or RMI JMX adapters using the [certsConfName](#) setting.

8. PKCS11 based certificates access

You can use PKCS11 based certificates access when using the JRE 5.0 only. You can use as server certificates with PKCS11, certificates located for example on smart cards.

To configure PKCS11, you'll have first to configure your JRE 5.0 to access the PKCS11 container. You'll have to create a config file (at least for Sun java PKCS11 impl) pointing to

JAFS SSL setup

the PKCS11 container lib :

```
# active card PKCS11 dll loc config file for
# sun.security.pkcs11.SunPKCS11 security provider
name = SmartCard
library = C:/WINNT/system32/acpkcs201.dll
```

And finally configure your `$JRE_HOME/lib/security/java.security` file to add a java security provider to access the PKCS11 compliant container. A complete guide can be found [here](#).

Once you're done with the settings you can define the PKCS11 keystore in the service config file as following:

```
<store-settings name="MyPKCS11Settings">
  <key-store password="123456" type="pkcs11" provider="SunPKCS11-SmartCard">
    <keyStoreFile>none</keyStoreFile>
  </key-store>
  ...
</store-settings>
```

The [keyStoreFile](#) tag value must be set to *none* and the [type](#) attribute to *pkcs11*. You can also specify an optional [provider](#) if you have configured multiple PKCS11 security providers.

Note that you can also configure PKCS11 access for cacerts store.

9. Client X509 certificate based user access

JAFS let you accept or reject users using SSL/TLS encrypted connection based on the data contained in the X509 certificate they provide.

The settings for this functionality are defined per server with the [trustedCertificates](#) configuration entry. Each [entry](#) setting will let you define what of data contained in the client certificate is accepted.

Any *entry* setting in the *trustedCertificates* set that match the client certificate data will let the user continue with the generic FTP authentication process (such as providing a username and password..). If no entry in the list is matching, the user will be disconnected.

```
...
<sslSettings isSSLServer="false" needCertificate="false"
  defaultProt="SSL" certsConfName="default">
  <trustedCertificates>
    <entry>
```

```

        <subjectDN caseSensitive="false">CN=HANS.* ,C=CH,O=.*</subjectDN>
    </entry>
    <entry>
        <subjectDN caseSensitive="false">CN=JOHN.* ,C=US</subjectDN>
        <issuerDN caseSensitive="true">CN=Verisign.*</issuerDN>
    </entry>
</trustedCertificates>
...
</sslSettings>
...

```

First, to enable such functionality you will need to turn the SSL [needCertificate](#) setting on to ask clients that connect to provide a **signed** X509 certificate.

Warning:

needCertificate set to on will **reject** all clients that do not provide a certificate signed by a root CA or by a key defined in the java trusted ca certs keystore.

Each entry is made of an [subjectDN](#) and/or [issuerDN](#) setting to define what certificate fields will be tested.

The values contained into those XML tags define the certificate values that needs to be tested. The data structure is defined by a set of certificate fields identifiers (O,CN,C...) delimited by commas, the values of the certificate fields that will be tested are compliant with regular expressions, finally the [issuerDN](#) or [subjectDN](#) caseSensitive settings instructs if the match is case sensitive or not:

```

<entry>
  <subjectDN caseSensitive="false">CN=HANS.* ,O=.*BI</subjectDN>
</entry>

```

Will match any certificate whose subject Common Name (CN) starts with *HANS* and whose Organization (O) ends with *BI*.

```

<entry>
  <subjectDN caseSensitive="false">CN=J.*N,C=CH</subjectDN>
  <issuerDN caseSensitive="true">CN=SomeRootCA</issuerDN>
</entry>

```

Will match any certificate whose subject Common Name starts with *J* and ends with *N*, whose Country (C) match *CH* and whose issuer Common Name match exactly *SomeRootCA*.

JAFS SSL setup

```
<entry>
  <issuerDN caseSensitive="false">CN=SomeOtherRootCA</issuerDN>
</entry>
```

Will match every certificates whose issuer Common Name match *SomeOtherRootCA*.

10. SSL/TLS cipher suites

You can define what kind of SSL/TLS cipher suites are allowed on the server, you'll simply need to add or remove [cipher](#) entries in the [enabledCipherSuites](#) list

An empty list allows all cipher implemented by the used Java Runtime Environment. If you set a wrong cipher name, a list of available cipher names for the current JRE will be printed in the error log and generic output console.

```
...
<sslSettings isSSLServer="false" needCertificate="false"
             defaultProt="SSL" certsConfName="default">
  ...
  <enabledCipherSuites>
    <cipher>TLS_RSA_WITH_AES_128_CBC_SHA</cipher>
    <cipher>SSL_RSA_WITH_3DES_EDE_CBC_SHA</cipher>
  </enabledCipherSuites>
  ...
</sslSettings>
...
```

The following setting will allows only client that implements the *TLS_RSA_WITH_AES_128_CBC_SHA* or *SSL_RSA_WITH_3DES_EDE_CBC_SHA* to connect.

11. CRL checks

For each keystore configuration blocks, you can load a defined set of Certification Revocation Lists (CRL). The lists are used to check during the peer SSL handshake if the provided certificate is contained into one of the certificates revocation list you have defined. When a match is found, the handshake is aborted and the session dropped. You can also define a period of time to refresh the lists to make sure that you always work with the latest ones. You can take a look [here](#) for additional info.

Warning:

CRL check adds some overhead during SSL handshake (80~90 ms for a check against a single list of 290Kb on a 2200 Mhz CPU). Make sure that you use only the required CRL for optimal performance.

```

...
<ssl-settings>
  <store-settings name="myStoresConfig">
    ...
    <crls reload-time="10">
      <crl location="http://www.someServer.com/someCRL.crl" name="MyFirstCRL"/>
      <crl location="http://www.someOtherServer.com/someCRL.crl" name="MyOtherCRL"/>
    </crls>
  </store-settings>
</ssl-settings>
...

```

12. Custom X509 trust managers

You can plug your own trust managers for X509 certificates. The managers will check the peer host certificate provided during the SSL session creation and let you accept or reject the session according to the certificate content. You can take a look at the [trustManager](#) config entry for more info about how to create your own trust manager.

```

...
<sslSettings isSSLServer="false" needCertificate="false"
  defaultProt="SSL" certsConfName="default">
  ...
  <customTrustManagers>
    <trustManager trustManagerFactory="net.sbbi.jafs.utils.HelloWorldFactory">
      <setting name="helloWorldValue">This is an top notch Hello World !</setting>
    </trustManager>
  </customTrustManagers>
</sslSettings>
...

```