SSL Configuration

Every communication passing through HTTP protocol can be secured with an SSL encryption. SSL encryption is enabled by default for communication between components of the monitoring platform (Middleware <-> Agent).

For web-based outputs (with a WEBSOCKET Class), communication can also be secured with SSL.

The Middleware application can enable SSL encryption for both web service listener and web-based outputs. In this case, the SSL configuration is shared between all of them.

It means there will be one single configuration that covers all these cases.

To enable SSL communication between monitoring platform components, go into the application configuration file and setup “**HTTPS**” value for **ProtocolServiceRequest** property.

Refer to Components configuration section for more information

To enable SSL communication for WEBSOCKET based outputs, go to the corresponding node in output section on the web-based GUI, and setup “**true**” value for **NetworkSecureSSL** property.

Refer to Output configuration section in user documentation, and refer to Outputs XML configuration file section in administration documentation for more information.

SSL configuration for web-based outputs is done in the same way as for the HTTPS communication protocols between components.

Both HTTPS communication and all potential web-based outputs instances share the same keystore file and certificate.

SSL keystore configuration is always located inside component “**etc**” folder and named “**ssl.properties**”. This file contains required information for webserver instances to initiate an SSL socket communication.

Just setup file content like following:

#####

keystore=etc/keystore.jks

password=password

**keystore** option must provide an absolute or relative path to a keystore file

**password** is both keystore and alias passwords

Password must be the same for keystore and certificate alias.

It is recommended to have only one alias for server’s certificate inside the keystore.

Note, the server’s certificate must cover all reachable URLs (hostnames) for the corresponding application.

Refer to Subject Alternative Name attribute documentation of X.509 certificates for more information.

A change on the SSL keystore configuration requires an application restart to take effect.

* + 1. Example of keystore file creation

In this section, we will cover all steps to enable SSL encryption for a particular server.

In the case the organization into which the certificate has to be deployed has its own PKI, usage of OpenSSL is not necessary. In this case Java **keytool** usage will be enough.

In the case you have to sign by yourself, server certificate with the Root CA key, and/or create your own Root certificate, then usage of OpenSSL is mandatory.

In real case, most of the times, servers have to be reachable from at least 2 DNS names. For instance, its short host name, and its FQND. The developed scenario will cover this situation as well.

Refer to Subject Alternative Name attribute documentation of X.509 certificates for more information.

It is recommended to use **keytool** application binary of the Java Runtime Environment used for the application execution. It is located under bin folder. Note the monitoring platform comes with its embedded Java version.

Consider “hostname” and “hostname.fqdn.com” as DNS aliases of the server.

Consider changing passwords and values of “CN”, “OU”, “O”, “L”, “S”, and “C” certificate attributes for both CA and server certificates.

Refer to **keytool** documentation for more information.

1 - Create keystore and certificate

keytool -genkeypair -keystore keystore.jks -dname "CN=hostname.fqdn.com, OU=IT, O=Company, L=City, S=Country, C=CountryAcronym" -keypass \*\*\*\*\* -storepass \*\*\*\*\* -keyalg RSA -alias server -validity 731 -ext SAN=dns:hostname,dns:hostname.fqdn.com -ext KU=digitalSignature,keyEncipherment -ext EKU=serverAuth

2 - Create a certificate request for signature to the certification authority

keytool -certreq -keystore keystore.jks -keypass \*\*\*\*\* -storepass \*\*\*\*\* -alias server -file hostname.csr

3 - In the case your organization has its own PKI, just send the CSR and wait for the signed server certificate.

In the case you want to manage or create a new PKI, go to 3-1 section for Root CA certificate creation, and server certificate signature with OpenSSL.

4 - Import provided Root CA and signed certificate of the server from certification authority

- Import of root CA:

keytool -import -alias cert1 -file root.pem -keystore keystore.jks -storepass \*\*\*\*\*

->> trust the certificate when prompted

- Eventually import other mid-certificate in the chain (repeat and change alias for all intermediate certificates in the chain)

keytool -import -alias cert2 -file sub.pem -keystore keystore.jks -storepass \*\*\*\*\*

- Finally import signed certificate

keytool -import -trustcacerts -alias server -file hostname.cer -keystore keystore.jks -storepass \*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Create Root CA and sign server certificate with OpenSSL

3-1 - Generate key for Root CA

openssl genpkey -algorithm RSA -out rootkey.pem -pkeyopt rsa\_keygen\_bits:4096

3-2 - Generate certificate CSR for Root CA self-signing

openssl req -new -key rootkey.pem -days 5480 -extensions v3\_ca -batch -out root.csr -utf8 -subj '/C=CountryAcronym/O=Company Root CA/OU=IT/CN=Company Root CA'

3-3 - Create an extension file (openssl.root.cnf) with following content

basicConstraints = critical, CA:TRUE

keyUsage = keyCertSign, cRLSign

subjectKeyIdentifier = hash

3-4 - Self sign Root CA certificate and append extensions

openssl x509 -req -sha256 -days 3650 -in root.csr -signkey rootkey.pem -set\_serial 1 -extfile openssl.root.cnf -out root.pem

3-5 - Sign server certificate request with Root key

openssl x509 -req -CA root.pem -CAkey rootkey.pem -in hostname.csr -out hostname.cer -days 731 -CAcreateserial

3-6 – Trust Root CA certificate

Eventually import “**root.pem**” Root CA certificate to your browser’s CA and/or into your Java Runtime Environment lists:

keytool -import -noprompt -trustcacerts -alias rootCA -file root.pem -keystore <JRE Path>/lib/security/cacerts -storepass changeit