Common Authentication Library Manual

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME
1.0.1	21 02 2019		\"UNICORE Team\"

Contents

Welcome to the documentation of the EMI X.509 Common Authentication Library, the Java edition!

This library was designed in the first place to support authentication and other X.509 PKI related operations of the components maintained by the EMI project. This document provides an overview and introductory material. It should be read accompanied by the library's API documentation in Javadoc format, available from the library project's web page.

1 Functional Description

The EMI X.509 Common Authentication Library, the Java edition, provides support for:

- off-line certificate validation,
- creation of SSL sockets (both server and client side),
- handling certificate DNs, also in text format,
- usage of proxy certificates: from proxy generation to validation.

The library provides implementation of an easy to use and flexible validation logic, which can take advantage of different trust material sources. Proxy certificates, which are commonly used in the grid environment, are fully supported. The library is designed in such a way, that it can be easily integrated with the standard JSSE stack. Therefore you can use it also in variety of 3rd party Java containers.

This documentation assumes at least a basic familiarity with the concepts of X.509 PKI, certificates and SSL. If you are new to these subjects, you have to improve your knowledge first.

2 Library user's guide

(aka Software Design Description)

The library classes can be found in variety of packages. All packages whose names start with `eu.emi.security.authn.x509.helpers` are considered an internal implementation. You can freely use them, but documentation might be less detailed and it is not guaranteed that the API won't change in future versions of the library. All classes in the remaining packages form the public, official API, which is intended for being used externally.

Each package provides a short description and classes are documented.

To understand how the library works it is necessary to be familiar with a few concepts:

Credentials
: Credentails are objects providing access to the real X.509 PKI identity information about an entity, i.e. its private key and certificate. Credentials are implementing the `eu.emi.security.authn.x509.X509Credential` interface. Credential objects are used in the first place to convert underlying credential representation to a format usable by the Java code, e.g. to create an SSL socket.

Validators and Trust Stores
: Validators are objects which validate provided certificate chains. The primary configuration of validation is a set of trusted issuers of certificates, called trust anchors or trusted Certificate Authorities (CAs). The set of trust anchors is called a trust store. Validators are tightly bound to the underlying trust stores and in fact the trust stores are usually the only difference between validators. However users of this library are using only the simple interface of validator, while underlying trust store is hidden. Validators are implementing the `eu.emi.security.authn.x509.X509CertChainValidator` interface. For example a concrete validator can use OpenSSL-style directory with certificates or a Java JKS file. Validators are used either directly to perform an off-line certificate chain validation or indirectly in SSL sockets to check peer certificates.

Creation of SSL sockets
: To make creation of SSL sockets easy there is a convenience class `eu.emi.security.authn.x509.impl.SocketFactoryCreator` which provides methods creating SSL server and client socket factories. Factory is created from credential and validator instances.

Proxy certificate generation
Library allows for creation of proxy certificates. It is possible to perform it in two ways. The simpler one is intended to be used when initial proxy certificate is created locally, from owned credentials. `eu.emi.security.authn.x509.proxy.ProxyCertificateOptions` object must be created with required parameters. Other approach can be also taken when proxy is generated for a service, which requested it by issuing a proxy certificate signing request (CSR). `eu.emi.security.authn.x509.proxy.ProxyRequestOptions` object is used to pass the required parameters. In both cases are handled by the proxy generation utility in the `eu.emi.security.authn.x509.proxy.ProxyGenerator` class.

Proxy certificate CSR generation
Library allows for creation of proxy CSR, what is typically needed on a service side, to request a proxy certificate from a service user. Proxy CSR can be populated with additional data like requested extensions. The tool for creating proxy CSRs is available in the class `eu.emi.security.authn.x509.proxy.ProxyCSRGenerator`. NOTE(!) that the party which is generating proxy certificate by signing the proxy CSR, can ignore the requested extensions. In this library the user has to extract requested parameters from the proxy CSR and if those are accepted, copy them to the `eu.emi.security.authn.x509.proxy.ProxyRequestOptions` object.

Utilities
There are also additional utilities in this library. It is important to note the following ones: `eu.emi.security.authn.x509.impl.X500NameUtils` provides methods to compare and print string representations of distinguished names; `eu.emi.security.authn.x509.impl.CertificateUtils` allows for loading and saving credentials and to pretty-print certificate chains. Classes `ProxyChainInfo`, `ProxyUtils` and `ProxyCSRInfo` from the `eu.emi.security.authn.x509.proxy` package allows programmers to inspect proxy certificates, chains with proxy certificates and proxy CSRs.

2.1 Library usage in a project

If a project using this library is managed by Maven, it is enough to add this library as POM's dependency, with the following snippet:

```
<dependency>
        <groupId>eu.emi.security</groupId>
        <artifactId>authnlib</artifactId>
        <version>DESIRED-VERSION-HERE</version>
</dependency>
```

The library is deployed in the Maven central repository.

Non-maven users must ensure that all *compile* dependencies of this library are available. The up to date (and very short) list of dependencies is available from the library documentation page. Choose in the leftmost menu: *Project Documentation* → *Project Information* → *Dependencies*. Note that relevant are only those dependencies which are enumerated in the first table as *compile* dependencies.

2.2 Note on usage of string representation of Distinguished Names

There are two commonly used formats for representing subjects and issuers of X.509 certificates - so called Distinguished Names, or DNs - as a string. Such names, also available in LDAP world, consist of sequence of attributes with values, defining an entity name.

It should be noted that full representation of a DN is a complex binary structure, which is stored in a certificate. Text representation must be human readable and all used representations loose some of the original DN data.

The first is the OpenSSL format, looking like */C=PL/DC=edu/CN=Some Person*. The second format is the RFC 2253 format, which encodes DNs in the following form: *CN=Some Person,DC=edu,C=PL*. This library uses, whenever possible, the RFC 22253 format, due to the following reasons: the RFC 2253 format is a standard, the OpenSSL format does not possess a formal specification and, what is worse, some of the DNs can be (en|de)coded ambiguously.

Another important issue related to DNs is their comparison. Attention must be paid not to compare text representations using literal string comparison. What is more, comparison using the standard JDK `javax.security.auth.x500.X500Prin`

cipal#equals(Object) method also can produce undesired results. **Therefore is is strongly suggested to use the `eu.emi.security.authn.x509.impl.X500NameUtils` class from this library to compare DNs whenever at least one of them is provided in the text format.**

2.3 Usage examples

2.3.1 Off-line certificate validation

Checking a certificate chain using OpenSSL style directory with trusted CA certificates:

```
/*
 * Validates toBeChecked chain using Openssl style truststore, from
 * the /etc/grid-security/certificates directory. Both kinds of
 * namespaces are checked and forced if are present. Truststore is
 * reread every minute. The additional settings are not defined and
 * so defaults are used: CRLs are forced if are present. Proxy
 * certificates are supported. No listeners are registered to
 * be notified about trusted CA certificates, CRLs or namespace
 * definitions reloading.
 */
X509Certificate[] toBeChecked = null;
X509CertChainValidator vff = new OpensslCertChainValidator(
        "/etc/grid-security/certificates",
        NamespaceCheckingMode.EUGRIDPMA_AND_GLOBUS, 60000);

ValidationResult result = vff.validate(toBeChecked);
if (result.isValid()) {
        //...
} else {
        List<ValidationError> errors = result.getErrors();
        //...
}
```

2.3.2 Creating SSLServerSocket using JKS keystore with trusted CA certificates

```
/*
 * A more complicated example. SSL sockets will be created with the
 * certificate validator from this library. It is configured to
 * trust all issuers from the provided JKS truststore.
 * Additionally two CRL sources are registered: one remote and
 * one local, using wildcard. CRLs are reloaded every hour and
 * remote CRLs are cached in /tmp/crls (useful if subsequent
 * download fails). Listener is registered which logs successful
 * and erroneous updates of the trust material.
 * Finally a local credential from another JKS file is loaded,
 * to be used as local side server's certificate and private key.
 */
char [] keystorePassword = "somePasswd".toCharArray(),
        ksPasswd = "passwd2".toCharArray(),
        keyPasswd = "passwd3".toCharArray();
String serverKeyAlias = "someAlias";
List<String> crlSources = new ArrayList<String>();
Collections.addAll(crlSources, "http://some.crl.distr.point1/crl.pem",
        "/etc/crls/*.crl");
StoreUpdateListener listener = new StoreUpdateListener() {
        public void loadingNotification(String location, String type,
                        Severity level, Exception cause)
        {
                if (level != Severity.NOTIFICATION) {
```

```
                        //log problem with loading 'type' data
                        //from 'location', details are usually in
                        //cause.
                } else {
                        //log successful (re)loading
                }
        }
};
CRLParameters crlParams = new CRLParameters(crlSources, 3600000,
        15000, "/tmp/crls");
ValidatorParamsExt commonParams = new ValidatorParamsExt(
        new RevocationParametersExt(CrlCheckingMode.REQUIRE,
                crlParams),
        ProxySupport.ALLOW, Collections.singletonList(listener));

KeystoreCertChainValidator v = new KeystoreCertChainValidator(
        "/my/truststore.jks", keystorePassword, "JKS", 1000,
        commonParams);

X509Credential c = new KeystoreCredential("/my/keystore.jks",
        ksPasswd, keyPasswd, serverKeyAlias, "JKS");
SSLServerSocketFactory sslSsf = SocketFactoryCreator.getServerSocketFactory(c, v);

ServerSocket sslSS = sslSsf.createServerSocket();
```

2.3.3 Checking if two DNs are equivalent

```
X509Certificate someCertificate = CertificateUtils.loadCertificate(
        inputStream, Encoding.PEM);
X500Principal dn1 = someCertificate.getSubjectX500Principal();
String dn2 = "CN=Bob,O=Example,C=EX";
//correctly compares binary DN with a string one
boolean equal = X500NameUtils.equal(dn1, dn2);
```

2.4 Best practices

- Validators are heavy-weight objects, consuming a lot of resources. The memory consumed is proportional to the size of the truststore. Additionally some of the validators start one thread to check for truststore updates. Therefore:

- always remember to call **dispose()** on validators which won't be used anymore,
- try to use as few instances as possible, typically one shared instance is enough.
  Of course all validators are thread safe.

2.5 Advanced topics

2.5.1 Low level validation API

Each `eu.emi.security.authn.x509.X509CertChainValidator` implementation offers possibility to register a `eu.emi.security.authn.x509.ValidationErrorListener`. Implementations of this listener are notified about each validation error found during validation. Typically the implementation can report the problem to the user, what is particularly useful when certificate validation is invoked indirectly (e.g. on SSL socket initialization) and actual error is buried deep in the exception stack.

Another, but less common, application of this interface is to influence validation logic. Listeners can ignore some of the errors or even change their description. **However this usage of the interface is a very hazardous choice.** The validator always tries

to provide a full list of validation errors, but it can not guarantee that all problems are found. Therefore programmer can not be 100% sure that after ignoring one of the errors no other could be found.

2.5.2 Registration of the BouncyCastle security provider

The library requires that the BouncyCastle security provider is registered in Java Virtual Machine. The provider is automatically registered if the public API of the library is used.

If the internal API (any class from any ***.helpers.*** package) is used directly, then the provider must be registered manually first. It can be done in any way suggested by the BouncyCastle library documentation or simply by invoking a utility method:

```
        CertificateUtils.configureSecProvider();
```

Note that the library registers the BouncyCastle provider only if it is not registered. Therefore you can modify the configuration of security providers (e.g. by moving the BouncyCastle provider to a specific position on the providers list) and be sure that your modifications will not be overwritten.

2.5.3 Validation errors

Library can produce many different errors related to certificate validation. All errors should provide meaningful for human explanation, category and code - see `ValidationError` class documentation for details. The codes are defined in `eu.emi.security.authn.x509.ValidationErrorCode`, however programmers may be also interested in codes' descriptions. Those are available in the source repository of the library, in the file: `src/main/resources/eu/emi/security/authn/x509/valiadationErrors.properties`

3 Building the library

To build the library Maven 2 is needed. After obtaining the source code (see the library main web page of coordinates) it is enough to run:

```
$ mvn package
```

to compile, test and package the library.

4 Limitations and known issues

- *limitation* The library doesn't allow for configuring PKIX path policy requirements, all verifications are performed with the standard settings (*any* policy). It is currently not decided if in future this limitation will be removed.
- *problem* Due to an error in the underlying BouncyCastle library the Proxy Address Restriction do not handle correctly masks which are NOT multiplies of 8 (/8, /16, /24, /32). This issue will disappear after updating dependencies to BC v. 1.47.