java_1_5_0_release_notes

Apache XML Security for Java 1.5.0 release notes

The 1.5.0 release is not binary compatible with the 1.4.x code due to extensive refactoring. The issues that have been fixed in the 1.5.0 release are available here. The release notes are as follows:

JDK 1.4.x support dropped

The 1.5.0 release drops support for JDK 1.4.x. The APIs have been updated to use generics where possible.

Xalan is no longer a required dependency

Xalan/Xerces/Xml-apis are now optional dependencies, and have been marked as such in the pom. These libraries are only required if you want to support the XPath here() function, which is not supported by the XPath API in the JDK.

GCM support added

Support for GCM algorithms has been added via a third-party JCE provider (e.g. BouncyCastle).

Support for RSA-OAEP key transport algorithms with strong digests

The 1.4.x releases only support using SHA-1 with the RSA-OAEP key transport algorithm for encryption and decryption. The 1.5.0 release supports stronger digests using the ds:DigestMethod child of xenc:EncrytionMethod. In addition, it fully supports the xenc11:MGF Algorithm as documented in the XML Encryption 1.1 specification.

JSR-105 provider has been renamed

The Santuario JSR-105 provider "org.jcp.xml.dsig.internal.XMLDSigRI" has been renamed as "org.apache.jcp.xml.dsig.internal.XMLDSigRI". This is to avoid conflict with the JDK provider of the same name. In addition, the XMLDSigRI provider's name has changed from "XMLDSig" to "ApacheXMLDSig".

Please note that it won't be possible to drop in the xmlsec.jar in the endorsed directory anymore and expect it to override the JDK's JSR 105 provider. You will need to edit the java.security file to register the new provider, or hard-code the provider name in your application code.

Secure Validation property

A new property has been added to enable "secure validation". This property is false by default. When set to true, it enforces the following processing rules:

- Limits the number of Transforms per Reference to a maximum of 5.
- Does not allow XSLT transforms.
- Does not allow a RetrievalMethod to reference another RetrievalMethod.
- Does not allow a Reference to call the ResolverLocalFilesystem or the ResolverDirectHTTP (references to local files and HTTP resources are forbidden).
- Limits the number of references per Manifest (SignedInfo) to a maximum of 30.
- MD5 is not allowed as a SignatureAlgorithm or DigestAlgorithm.
- Guarantees that the Dereferenced Element returned via Document.getElementById is unique by performing a tree-search.

This functionality is supported in the core library through additional method signatures which take a boolean, and in the JSR-105 API via the property "org.apache.jcp.xml.dsig.secureValidation, e.g.:

```
XMLValidateContext context = new DOMValidateContext(key, elem);
context.setProperty("org.apache.jcp.xml.dsig.secureValidation", Boolean.
TRUE);
```

Dynamic registration of default algorithms

The 1.4.x release loads all supported algorithms/implementations by reading and parsing an xml configuration file. This gives the user the power to override the default configuration file by setting the appropriate system property, and to add/remove algorithms as desired.

The downside to the current configuration approach is the performance issue of reading a file from the filesystem, parsing it, and then class-loading all of the implementations. Given that the majority of users probably just use the default algorithms that come with the library, this represents a performance issue on start-up.

In 1.5.0, the default algorithms are loaded dynamically. To revert to the older behaviour of loading algorithms from a properties file then set the System property "org.apache.xml.security.resource.config" to point to the file.

Major changes to how Elements are resolved

In the 1.4.x releases, the IdResolver class is responsible for retrieving Elements for a given Id. In the 1.5.0 release it will no longer attempt to search the current Document for an Element matching the requested Id in certain "known" namespaces (e.g. SOAP Message Security). Instead, the ResourceResolver implementations that are responsible for resolving same-Document URI references resolve Elements by querying Document.getElementById(). The IdResolver has been deprecated, so that it is no longer possible to register an Element by Id, and the "get" methods simply delegate to the DOM call above.

So what does this mean for the user? The user is now responsible for registering any same-Document Elements by Id, which must be resolved as part of the signature creation/validation process. This can be done in two ways:

- Using the DOM APIs: Element.setIdAttribute*
- Using the JSR-105 API: DOMCryptoContext.setIdAttributeNS

Better protection against Signature Wrapping Attacks

A Signature Wrapping attack can occur when a malicious entity duplicates some signed Element in a document, and then modifies the content of the duplicated Element, but keeps the same Id. If the signature validation process only finds the initial Element, then signature validation will pass, and the user might be fooled into thinking that the modified Element has been signed, as it has the same Id as the original validated Element.

The implication of this attack is that it is vital that the user checks that the Elements that were signed correspond to the Elements that he/she expects to be signed. In other words, the Elements that were signed should be located in a specific "known" place in the Document. The best way of facilitating this check is to make sure that the signature validation process returns the Elements that it validated. The JSR-105 API supports this via the "javax.xml.crypto.dsig.cacheReference" boolean property, that can be set on the context.

However, the older XMLSignature, which pre-dates the JSR-105 API, did not include this functionality. This has been fixed in the 1.5.0 release, e.g.:

```
XMLSignature signature = ...
// Perform validation and then...
SignedInfo signedInfo = signature.getSignedInfo();
for (int i = 0; i < signedInfo.getLength(); i++) {
    Reference reference = signedInfo.item(i);
    ReferenceData refData = reference.getReferenceData();
    ....
}
```

ReferenceData is a new interface that duplicates the way the JSR-105 API returns references. Three different implementations have been provided depending on whether the dereferenced Element is a NodeSet, Element or OctetStream.

Finally, some new functionality has been added when secure validation is enabled, to ensure that when an Element is de-referenced via Document.getElementById() (as described above), no other Element in the tree has an Attribute that is also registered as an Id with the same value. This is done via a tree search, and guarantees that the Element retrieved via the getElementById call is unique in the Document (something that is not guaranteed by the contract of getElementById). However note that another Element could still exist in the tree with a matching Id in another namespace.