wolfSSL JNI Manual

December 4, 2015, version 1.4.0

Introduction

The wolfSSL JNI package provides a Java interface to the wolfSSL SSL/TLS library, providing Java applications with SSL/TLS support up to the current TLS 1.2 and DTLS 1.2 standards. The interface is provided through the use of JNI and standard Java practices. In addition to the interface, the package provides an example client and server, written in Java, which utilize the interface to make an SSL/TLS connection.

As this interface wraps around the native wolfSSL library, this document should be used in reference together with the wolfSSL Manual:

https://wolfssl.com/wolfSSL/Docs-wolfssl-manual-toc.html

Table of Contents

Getting wolfSSL JNI Source Code

The most recent version of wolfSSL JNI can be downloaded from the wolfSSL website as a ZIP file from the Download page:

https://wolfssl.com/wolfSSL/download/downloadForm.php

Requirements

To compile and use this interface, users must have Java installed on the development machine. Prior to compiling the JNI wrapper, the wolfSSL library must be compiled and installed in a location which can be found by the JNI wrapper. This release of the Java wrapper has been tested against wolfSSL 3.7.0.

The package utilizes the ant build system to make compilation, testing, and documentation easier and more streamlined. As such, if users wish to use the existing ant build script, ant will need to be installed and available on the development machine.

JUnit is used for unit tests. JUnit will need to be installed on the development machine in order for unit tests to work. Users can download JUnit from http://www.junit.org.

After downloading junit-x.x.jar and hamcrest-core-x.x.jar from the JUnit website, you may need to add them to your classpath. For example, on Linux this would be similar to:

```
$ export
CLASSPATH=$CLASSPATH:/usr/share/java/junit-4.12.jar:/usr/share/java/hamcrest-co
re-1.3.jar
```

When running JUnit tests, this package will look for junit-4.12.jar and ant-junit4.jar at the following locations:

```
$JUNIT_HOME/junit-4.12.jar
$JUNIT_HOME/hamcrest-core-1.3.jar
$JUNIT_HOME/ant/ant-junit4.jar
```

Please set the $JUNIT_HOME environment variable to match your system where the above files are located, ie:

```
$ export JUNIT_HOME=/usr/share/java
```

Package Design

The Java interface package has the following directory structure.

<package_root>		
build.xml		(ant build script)
/docs		(javadocs)
/examples		(examples)
	/certs	(test certs/keys)
java.sh		(JNI gcc script)
/lib		(output directory for compiled JNI lib and JAR file)
/native		(native JNI code)
/src		
	/java	(Java sources)
	/test	(test code)

Building

wolfSSL will need to be compiled and installed on the development machine using the following build options:

```
$ cd wolfssl-3.7.0
$ ./configure --enable-jni
$ make
$ sudo make install
```

To compile the JNI interface, run the following commands from the root package directory.

`$ ./java.sh`	(compiles native JNI sources into shared library)
`$ ant`	(compiles Java sources, javadocs, JNI headers, JAR, and runs available tests)

The java.sh script tries to detect where Java is installed on your system, using the javaHome variable in java.sh to store this path. If your system has issues finding the Java installation path, manual edit of this variable may be necessary in java.sh.

Cleanup

To clean the package, run the following commands from the root directory:

`ant clean`	(cleans all Java sources and .JAR)
`ant cleanjni`	(cleans native object files and shared library)

Examples

This interface includes an example server and client written in Java to provide developers with a simple usage example. The example client and server are located in the <package_root>/examples directory. Each is accompanied by a wrapper script. The wrapper script should be used when running the programs to more easily set the classpath and library path options.

The examples can be run from the root package directory using:

```
./examples/server.sh
./examples/client.sh
```

Each example provides several command line options which can be used to customize how the example(s) run. Use the "-?" option to show available command line options:

```
Java example server usage:
-?          Help, print this usage
-p <num>    Port to connect to, default 11111
-v <num>    SSL version [0-3], SSLv3(0) - TLS1.2(3)), default 3
-l <str>    Cipher list
-c <file>   Certificate file,        default ../certs/client-cert.pem
-k <file>   Key file,                default ../certs/client-key.pem
-A <file>   Certificate Authority file,    default ../certs/client-cert.pem
-d          Disable peer checks
-s          Use pre shared keys
-u          Use UDP DTLS, add -v 2 for DTLSv1 (default), -v 3 for DTLSv1.2
-iocb       Enable test I/O callbacks
-logtest    Enable test logging callback
-o          Perform OCSP lookup on peer certificate
-O <url>    Perform OCSP lookup using <url> as responder
-U          Atomic User Record Layer Callbacks
-P          Public Key Callbacks
-m          Enable CRL directory monitor
```

The examples demonstrate usage of several things, including using custom I/O callbacks, a custom verify callback, DTLS cookie generation callback, Atomic Record callbacks, public key (ECC, RSA) callbacks, and PSK support.

Sources for the example client can be found in <package_root>/examples/Client.java and the example server in <package_root>/examples/Server.java. Custom I/O callbacks can be found in MyRecvCallback.java and MySendCallback.java, with the I/O context found in MyIOContext.java. The custom verify callback can be found in VerifyCallback.java. Examples of atomic user record layer callbacks and public key callbacks are also included in the <package_root>/examples directory.

The verify callback currently returns 1 (failure), but does no processing of the peer certificate chain. A real-world application would want to do any desired certificate verification processing before returning failure or success.

Currently DTLS usage requires custom I/O callbacks and DTLS cookie generation callback to be registered. Examples of these can be found in Server.java and Client.java.

API Documentation

The ant build system compiles up-to-date Javadocs when run. To view the most current Javadocs for the package, open the local javadocs index file in a web browser.

<package_root>/docs/index.html

or browse to the online version at:

http://www.wolfssl.com/documentation/wolfssl-jni-javadocs/index.html

License

Like wolfSSL (wolfSSL), this package is dual licensed under both the GPLv2 as well as a standard commercial license. Full license details can be found on the wolfSSL Licensing page. The GPLv2 license header included in the wolfSSL JNI download package is copied below:

Support

Please send questions or comments to wolfSSL at **support@wolfssl.com**.

References

wolfSSL Product Page: https://wolfssl.com/wolfSSL/Products-wolfssl.html
wolfSSL Documentation: https://wolfssl.com/wolfSSL/Docs.html
wolfSSL Manual: https://wolfssl.com/wolfSSL/Docs-wolfssl-manual-toc.html