MIT Kerberos

Kerberos Java GSS-API Wrapper

November 19th 2012, version 1.0

Introduction

This package provides a Java GSS-API wrapper around the the MIT Kerberos GSS-API native library. This wrapper conforms to the GSS-API Java bindings via RFC 5653. One of the main goals of this project is to bring GSS-API functionality to the Android platform, which previous to this project lacked both Kerberos and GSS-API support. Using this project, Android developers are able to use GSS-API functionality in their Android NDK applications.

For a working example of an Android NDK application using this Java GSS-API interface, please reference the "Kerberos Android NDK" project hosted on GitHub: https://github.com/cconlon/kerberos-android-ndk. This project provides a sample Android NDK application showing how to use MIT Kerberos and GSS-API functionality in an Android application.

Table of Contents

1. Requirements

You must have SWIG installed on your development machine in order to build this GSS-API wrapper. The Java GSS-API bindings are wrapped around a native SWIG-generated layer that then in turn interfaces to the native Kerberos GSS-API library. To download and install SWIG, please see the project homepage at http://www.swig.org/. This project has been developed using SWIG version 1.3.40 running on Linux.

To use this interface in your Android NDK application, you need to include cross-compiled versions of the MIT Kerberos libraries for Android in your project. For details about these libraries and an example of how to include them in your project, please see the "Kerberos Android NDK" application on GitHub: https://github.com/cconlon/kerberos-android-ndk for both pre-compiled libraries and instruction on how to compile them manually.

If you want to rebuild the pre-built Kerberos libraries, please use the android-config.sh shell script found in the above noted project. This will setup the correct autoconf environment for the MIT Kerberos libraries to be cross-compiled for the Android platform. More detailed instructions can be found in the script comments.

The development machine must also have a working Java implementation installed in order to compile the source and examples.

2. Project Design

This project is composed of several layers - most of which are invisible to the end Java API user. The individual layers are visualized in the following figure. The native MIT GSS-API library is first wrapped using SWIG to form a temporary C/Java layer (LAYER 1). This first layer may be used directly, but is more tedious and less standardized than the org.ietf.jgss interface. A top-level Java API is then wrapped around the SWIG-generated layer. This top-level Java API (LAYER 2) conforms to the org.ietf.jgss interface specification. The interface is located in the org.ietf.jgss package while the implementation is located in the edu.mit.jgss package.

Layer 2: Java GSS-API interface (org.ietf.jgss, edu.mit.jgss)
        |-----> **Layer 1:** SWIG-generated interface layer handling C-Java interaction
                |-----> Native MIT Kerberos/GSS-API libraries

2.1 Package Contents:

A short description of the main file and directory structures in this package are below.

gsswrapper.i

This is the SWIG interface file. It contains all of the code and typemaps needed by SWIG to generate the corresponding "LAYER 1" Java interface for the MIT GSS-API library.

gsswrapper_wrap.h

This is a header file that contains function prototypes for the SWIG-generated C wrapper functions. If functions are changed in gsswrapper.i, this file should be updated to match accordingly.

src/java/edu/mit/jgss/swig

Location of the SWIG-generated .java files which provide "LAYER 1" Java access to the native MIT GSS-API library using a similar interface to GSS-API C bindings.

src/java/edu/mit/jgss

Location of the "LAYER 2" MIT Java GSS-API implementation of RFC 5653.

src/java/org/ietf/jgss

Location of the Layer 2 Java GSS-API interface files as outlined in RFC 5653.

3. Building

The Java GSS-API interface can currently be built for use on a standard desktop environment, or integrated into an Android NDK project.

3.1 Desktop Environment

To build the GSS-API interface and examples on a desktop environment,

1. Change directory (cd) to the root package directory and edit the **JavaBuild.sh** file to match your system's Java and Kerberos configuration.

NOTE: If building on OS X, the Java include directory will most likely be something similar to:

/System/Library/Frameworks/JavaVM.framework/Versions/A/Headers

For OS X, you'll also need to change the extension of the shared library being compiled by SWIG to .dylib (libgsswrapper.dylib) instead of .so which is used by standard Linux environments.

If building on Linux, the Java include directory may be similar to:

/usr/lib/jvm/java-6-openjdk/include

2. Verify that the library being loaded by the SWIG wrapper (gsswrapper.i) is the correct one. This should be the name of the library being created by the JavaBuild.sh script.

3. Run ./JavaBuild.sh

   This will run the correct SWIG command, create libgsswrapper.so (.dylib) as well as generate and compile all the necessary JNI and Java files needed for the interface and examples.

4. You may need to set **LD_LIBRARY_PATH** to include paths to the MIT Kerberos libraries on your system as well as the location of the SWIG-generated library. For example, something similar to:

   Linux:
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib:
   /home/myuser/kerberos-java-gssapi/

   Mac:
   export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:/usr/local/lib:
   /Users/myuser/kerberos-java-gssapi/

5. Build the .java source files using the ant build system with the following command (issued from the package root directory). Compiled .class files will be placed into the ./build and ./examples/build directories.

   ant

6. To easily clean up the package and return it to its original state, run the following commands from the package root directory. This will delete all compiled Java source files and the SWIG-generated .java files located in src/java/edu/mit/jgss/swig and native/.

ant clean
ant cleanswig

3.2 Android NDK Environment

These instructions assume that you already have an Android NDK application created - or at least the directory structure of the application setup.

1. Copy both **gsswrapper.i** and **gsswrapper_wrap.h** into your project's 'jni' directory.

2. Create the following directory structures under your NDK application project root directory:

   src/edu/mit/jgss
   src/edu/mit/jgss/swig
   src/org/ietf/jgss

3. Copy the contents of the following src/java folders to their equivalent in your Android NDK application:

   src/java/edu/mit/jgss -> src/edu/mit/jgss
   src/java/org/ietf/jgss -> src/org/ietf/jgss

   These folders contain the Java GSS-API (LAYER 2) interface.

4. If needed, verify that the library being loaded by the SWIG wrapper (gsswrapper.i) is the correct one. For an Android NDK library, this will be the name of your native shared library. You may also need to load the native shared library into your application code manually, as demonstrated by the example NDK application in the https://github.com/cconlon/kerberos-android-ndk project.

5. From your Android NDK project's root directory, run the following command. This will generate all the necessary SWIG interface files.

   swig -java -package edu.mit.jgss.swig -outdir ./src/edu/mit/jgss/swig
   -o ./jni/gsswrapper_wrap.c ./jni/gsswrapper.i

6. Make sure you have included cross-compiled versions of the necessary MIT Kerberos libraries as well as the Kerberos and CyaSSL header files in your Android NDK project. See the "kerberos-android-ndk" sample app (referenced above) for an example and instructions on how to cross compile the Kerberos libraries. A common place to put these would be in ./jni/libs and ./jni/include, as done in the example Kerberos NDK project.

7. Add necessary MIT Kerberos libraries and gsswrapper_wrap.c to the Android.mk file as shown in the example Kerberos Android application. This may vary depending on how your project is set up.

8. In your application .java files, add an import for the org.ietf.jgss package:

   import org.ietf.jgss.*;

9. Build and install your NDK application like normal (ndk-build, ant debug, ant debug install, etc.). For more details, please see the README included in the kerberos-android-ndk project. You may also need to push certain configuration and keytab files to your device prior to using the GSS-API layer.

4. Examples

To help in understanding how to use this Java GSS-API wrapper in your application or project, there are two sets of client and server examples included in this package. The first one is set of example client/server applications which directly use the raw SWIG, LAYER 1, interface. The second set of client/server examples use the more standardized Java GSS-API (org.ietf.jgss) LAYER 2 interface.

Both sets of example make use of a utility class, Util.java for the LAYER 1 examples and GssUtil.java for the LAYER 2 examples. It is recommended to use the LAYER 2 examples - as they demonstrate programming and API usage that is more common in the Java programming language.

Before running these examples, your krb5.conf file and Kerberos KDC need to be set up correctly to match the principal names you will use in the examples.

LAYER 1 EXAMPLES

4.1 Client (examples/client.java)

The client example expects that the user has already run kinit to receive a TGT for the client principal. The client principal name, server service name, server address, and the server port need to be set to the correct values in client.java before it is compiled.

When running the client in a desktop environment, after starting the example server (server.java), run:

./examples/client.sh

from the root kerberos-java-gssapi directory. After the client has been started, it will do several things, including:

1. Establish a GSSAPI context with the example server
2. Sign, encrypt, and send a message to the server using gss_wrap.
3. Verify the signature block returned by the server with gss_verify_mic.
4. Repeat steps (2) and (3) but using gss_seal / gss_verify.
5. Perform misc. GSSAPI function tests

4.2 Server (examples/server.java)

The server example is designed to be run on a standard desktop environment and has been tested on Linux. It is located in server.java and will connect to the example client (client.java). The server and server service name should be modified to the desired values before compiling the example.

On a desktop environment, after configuring and compiling the server, start it using:

./examples/server.sh

from the root kerberos-java-gssapi directory. Once started, the server will wait for a client connection. When a connection is received, the server will do several things, including:

1. Establish a GSS-API context with the example client
2. Receive and unwrap a wrapped message from the client
3. Generate and send a signature block for the received message.

LAYER 2 EXAMPLES

4.3 Client (examples/gssClient.java)

For Android, the functionality of this client application has been built into the Android NDK sample application (KerberosAppActivity.java) in the kerberos-android-ndk repository. The client is also provided as a standalone Java example which may be run on standard desktop operating environments.

The client example expects that the user has already run kinit to receive a TGT for the client principal. The client principal name, server service name, server address, and server port need to be set to the correct values in gssClient.java before compilation.

When running the client in a desktop environment, after starting the example server (gssServer.java), run:

./examples/gssClient.sh

from the kerberos-java-gssapi root directory. After the client has been started, it will do several things, including:

1. Establish a GSSAPI context with the example server.

2. Sign, encrypt, and send a message to the server.
3. Verify the signature block returned by the server.

4.4 Server (examples/gssServer.java)

The server example is designed to be run on a standard desktop environment and has been tested on Linux. It is located in gssServer.java and will connect to either the example GSS-API client (gssClient.java) or the example Kerberos Android NDK application (kerberos-android-ndk). The server service name should be modified to the desired value before compiling and running the example.

On a desktop environment, after configuring and compiling the server, start it using:

./examples/gssServer.sh

from the root kerberos-java-gssapi directory. Once started, the server will do several things, including:

1. Establish a GSS-API context with the example client.
2. Unwrap a signed and encrypted message that the client sends.
3. Generate and send a signature block for the received message.

Before starting the example server, there should be an entry in your system keytab for the server service principal.

5. Notes

1. Because the org.ietf.jgss package already exists on most desktop Java installations, it is necessary to set the bootclasspath variable when running applications built with MIT's org.ietf.jgss package. To see an example of this, please reference the test example scripts in ./examples (ie server.sh, client.sh).

   When using the bootclasspath, the MIT org.ietf.jgss classes are treated as system classes. All system classes will only lookup shared libraries in $JAVA_HOME/bin. As such, one of several actions needs to be done when using this package with bootclasspath:

   a. Install gsswrapper.so library into $JAVA_HOME/bin
   b. Set the sun.boot.library.path to include the gsswrapper.so library path. The syntax is:

      java -Dsun.boot.library.path=$JAVA_HOME/bin:/path/to/gsswrapper.so

   The need to use bootclasspath should not be necessary on Android, as the platform doesn't have an existing org.ietf.jgss package installed.

2. The Layer 2 MIT GSS-API Java implementation does not have support for the SPI framework that is specified in RFC 5653. This framework is specified as optional in the RFC. The current edu.mit.jgss package wraps directly around the Layer 1 SWIG Java GSS-API package (edu.mit.jgss.swig).

6. SWIG Interface Details

Note that this section explains the details of the SWIG, LAYER 1 interface. For notes on the Java Bindings (org.ietf.jgss) interface, please refer to Section 7.0 of this document.

The SWIG, LAYER 1, Java GSS-API interface functions are located in gsswrapper.java, while there are several separate Java files for each GSS-API structure. Because Java is an object oriented language, the Java GSS-API interface usage differs slightly from the standard MIT GSS-API usage. Java doesn't make use of pointers as the C language does. Because of this, each GSS-API structure has been standardized to a single object in Java, following the naming scheme XXXX_desc, where XXXX is the name of the structure (ex: The Java object for a gss_OID object is gss_OID_desc).

For example, in the native MIT GSS-API library, there can be direct structure usage, a pointer to that structure, or a pointer to a pointer. In Java, all of these usages are simplified to a single object. The SWIG wrapper is responsible for converting this object into the correct C form to pass back to the native MIT GSS-API library.

In Java, GSS-API functions can be accessed through the gsswrapper.java file like so:

```
maj_status = gsswrapper.gss_accept_sec_context(min_status, ...);
```

Because many of the native GSS-API functions return values inside function parameters as OM_uint32 types, you must use a Java array for those parameters. For example, defining the GSS-API min_status variable in Java can be done like so:

```
long[] min_status = {0};
```

In this case, the returned min_status value by the gss_accept_sec_context method will be placed into the first element of the Java long[] (min_status[0]).

6.1 GSS-API Java objects

GSS-API structures/objects can be created like normal Java objects. For example to create a gss_OID and a gss_cred_id_t object, you would use:

```
gss_OID_desc myoid = new gss_OID_desc();
gss_cred_id_t_desc mycreds = new gss_cred_id_t_desc();
```

A list of GSS-API objects which are available through the Java interface include:

gss_OID_desc() / gss_OID_desc(String mechanism)
        mechanism = optional mech string to initialize the OID with
        (ex: "{ 1 2 840 113554 1 2 2}").
gss_OID_set_desc()
gss_buffer_desc() / gss_buffer_desc(String value)
        value = optional string to initialize the buffer with.
gss_channel_bindings_struct()
gss_cred_id_t_desc()
gss_ctx_id_t_desc()
gss_name_t_desc()

6.2 GSS-API Constants

All GSS-API constants, including calling errors, routine errors, supplementary info bits, etc, are located in the gsswrapperConstants.java file. If your application code implements this file, the constants can be used directly. For example, if a Java class implements gsswrapperConstants:

```
public class myClass implements gsswrapperConstants
{
    ...
}
```

Then the constants can be used directly inside of that class:

```
if (maj_status != GSS_S_COMPLETE) { ... }
```

6.3 Status Code Macros

GSS-API macros that test status codes for error conditions are located in gsswrapper.java. Specific details can be seen in either gsswrapper.java or gsswrapper.i. The macros included are:

GSS_CALLING_ERROR
GSS_ROUTINE_ERROR
GSS_SUPPLEMENTARY_INFO
GSS_ERROR
GSS_CALLING_ERROR_FIELD
GSS_ROUTINE_ERROR_FIELD
GSS_SUPPLEMENTARY_INFO_FIELD

6.4 Helper Functions

[gss_display_status_wrap]

This method has been included in the GSS-API Java interface to provide a wrapper around the standard GSS-API gss_display_status function. It is needed because Java passes in a long for min_status instead of a pointer to an OM_uint32. The Java prototype for this method is:

```
public static long gss_display_status_wrap(
        long min_status,
        long status_value,
        int status_type,
        gss_OID_desc mech_type,
        long[] message_context,
        gss_buffer_desc status_string);
```

[getDescArray]

This method allows the easy retrieval of gss_buffer_t value by Java. In the example Kerberos Android application it is used to get the value of an outputToken (gss_buffer_desc) as a Java byte[] to send across the network. The Java prototype for this method is:

```
public static byte[] getDescArray(gss_buffer_desc buffer);
```

[setDescArray]

This method allows the value of a gss_buffer_t object to be set from Java using a Java byte[] as input. In the example Kerberos Android application, it is used to set the value of an inputToken after it has been received as a byte[] from the example server. The Java prototype for this method is:

```
public static int setDescArray(gss_buffer_desc buffer, byte[] javaArray);
```

6.5 Structure Extensions

[gss_OID_set_desc.getElement]

This method is used to get a specific member of a gss_OID_set_desc object. It takes an offset as input and returns a gss_OID_desc object. The Java prototype is:

```
public gss_OID_desc getElement(int offset);
```

[gss_OID_desc.equals]

This is a comparison function for a gss_OID_desc and input mech string. It returns 1 if the two are equal, otherwise 0. The Java prototype is:

public int equals(String mechString_in);

6.6 GSS-API Methods

The following standard GSS-API functions are included in the Java GSS-API interface. For documentation regarding each individual function, please reference standard GSS-API documentation.

gss_acquire_cred
gss_release_cred
gss_init_sec_context
gss_accept_sec_context
gss_process_context_token
gss_delete_sec_context
gss_context_time
gss_get_mic
gss_verify_mic
gss_wrap
gss_unwrap
gss_display_status (wrapped as gss_display_status_wrap)
gss_indicate_mechs
gss_compare_name
gss_display_name
gss_import_name
gss_release_name
gss_release_buffer
gss_release_oid_set
gss_inquire_cred
gss_inquire_context
gss_wrap_size_limit
gss_add_cred
gss_inquire_cred_by_mech
gss_export_sec_context
gss_import_sec_context
gss_release_oid
gss_create_empty_oid_set
gss_add_oid_set_member
gss_test_oid_set_member
gss_str_to_oid
gss_oid_to_str
gss_inquire_names_for_mech

gss_inquire_mechs_for_name
gss_sign
gss_verify
gss_seal
gss_unseal
gss_export_name
gss_duplicate_name
gss_canonicalize_name
gss_pseudo_random
gss_store_cred
gss_set_neg_mechs
gss_indicate_mechs_by_attrs
gss_inquire_attrs_for_mechs
gss_display_mech_attr
gss_inquire_saslname_for_mech
gss_inquire_mech_for_saslname

7. Java GSS-API Details (org.ietf.jgss)

The LAYER 2 Java GSS-API interface was designed to make it easier for Java developers to use the native MIT GSS-API library. The GSS-API interface conforms to the org.ietf.jgss package outlined in RFC 5653. There are a few things to take note of - please see section 5.0 of this document for more details.

As the org.ietf.jgss interface is fairly well documented, there isn't extensive API documentation located in this document. Please refer to standard Java GSS-API org.ietf.jgss documentation such as the Oracle, OpenJDK, or Apache Harmony documentation.

8. Licenses

9. Support

If you have any questions or comments, please post to the krbdev mailing list (http://web.mit.edu/kerberos/mail-lists.html) or contact support@yassl.com.

10. References

MIT Kerberos: http://web.mit.edu/kerberos/
yaSSL: http://www.yassl.com/

Kerberos Java GSS-API Wrapper: https://github.com/cconlon/kerberos-java-gssapi
Example GSS-API Android NDK App: https://github.com/cconlon/kerberos-android-ndk

RFC 5653: http://tools.ietf.org/html/rfc5653