Kerberos Programming on Windows

By Weijun on May 24, 2011

This article was published as http://java.sun.com/javase/6/docs/technotes/guides/security/kerberos/jgss-windows.html some time in 2009, but the original link does not exist anymore. It's copied here mainly for archive purpose and a lot of thing have changed since. I might or might not update it.

This article talks about Kerberos programming on Windows, especially in a Kerberos environment of Windows Active Directory (AD), with all clients and services running on Windows platforms in AD domains. The typical client/server environment described here is Windows XP and Windows Server 2003. We may talk about other scenarios if necessary.

Note: Kerberos programming in Java is done through the JGSS-API. Please make sure you're familiar with basic JGSS concepts and programming styles. Read the Java SE documentation of JGSS section first.

Basic Setup

There are three roles in Kerberos: the KDC, the client, and the server. We're talking about writing Java programs on the client or the server, or both.

First, you must have a Windows Active Directory server running, and all clients and servers joining this AD domain. In order for Java to recognize this environment, extra configurations are needed on both the client and the server side.

Realm and KDC Info

There are two ways to inform a Java program what the Kerberos realm and KDC are:

1. krb5.ini configuration file
    - krb5.ini should contain the realm info and hostname of the KDC for this realm. For example:

      ```
      [libdefaults]
      default_realm = MY.REALM
      realms]
      MY.REALM = {
          kdc = kdc.my.realm
      }
      ```

    - The file location can be specified by system property java.security.krb5.conf. Otherwise, Java will try to locate this file in these locations (ordered by):
        1. %JAVA_HOME%/lib/security/krb5.conf
        2. %WINDOWS_ROOT%/krb5.ini
2. System properties java.security.krb5.realm and java.security.krb5.kdc.

Please note that these two configurations cannot be provided at the same time.

In JDK 7, when neither of the two ways above is used. Java will try to read the realm and KDC settings from Windows environment variables.

JAAS login config file

Since JGSS uses JAAS to acquire the initial Kerberos credentials, a JAAS login config file is always needed. The location of this file should be specified inside the java.security file or using the system property java.security.auth.login.config. Read here for details.

The login module required here is com.sun.security.auth.module.Krb5LoginModule, we'll talk about the details in later sections for the client side and the server side respectively.

In JDK 7, when no JAAS login config file is specified, pre-defined entries are created for the client side and server side respectively:

For the client side:

```
com.sun.security.jgss.krb5.initiate {
    com.sun.security.auth.module.Krb5LoginModule
    required
    useTicketCache=true
```

```
    doNotPrompt=false
};
```

For the server side:

```
com.sun.security.jgss.krb5.accept {
    com.sun.security.auth.module.Krb5LoginModule
    required
    useKeyTab=true
    storeKey=true
    doNotPrompt=true
    isInitiator=false;
};
```

TGT accessibility

By default, Windows does not allow the session key of a TGT to be accessed. Please add the following registry key on the client side, so that the session key for TGT is accessible and Java can use it to acquire additional service tickets.

For Windows XP and Windows 2000, the registry key and value should be:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos
Value Name: allowtgtsessionkey
Value Type: REG_DWORD
Value: 0x01
```

For Windows 2003 and Windows Vista, the registry key and value should be:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters
Value Name: allowtgtsessionkey
Value Type: REG_DWORD
Value: 0x01
```

Programming the client side

The Initial Credentials

JGSS uses JAAS to get the initial credentials (in the case of Kerberos, the initial TGT). Java tries to get it in this order:

1. File credentials cache (%HOME%\krb5cc_userid for Windows)
2. Native credentials cache (LSA, or Local Security Authority, for Windows)
3. Read key from a keytab file and use AS_REQ to acquire credentials from KDC
4. Prompt for username and password and use AS_REQ to acquire credentials from KDC

Not all of them will be tried. The actual behavior depends on what's specified in Krb5LoginModule of your JAAS long config file:

- If useTicketCache=true, 1, 2 will be tried
- If useKeyTab=true, 3 will be tried
- If doNotPrompt=true, 4 will not be tried

Note that all of these parameters' default values are false. They can be specified in any combination.

The most common case on a Windows client is that the user has already logged on to the system as an AD account, which means there's a native credential cached in LSA. This goes the 2nd way above. However, if the client platform is not Windows, or, although it's Windows but the user is not logged on as an AD account, there's no LSA cache available. Please check the availability of an LSA cache using the MS klist.exe tool (Attention: not the klist.exe comes with Java) or kerbtray.exe (for GUI lovers) provided by Microsoft.

Therefore, the typical JAAS login config file for client should look like this:

```
com.sun.security.jgss.krb5.initiate {
    com.sun.security.auth.module.Krb5LoginModule
    required
    useTicketCache=true
    doNotPrompt=false
};
```

This means the ticket cache (LSA) should be used automatically, doNotPrompt=false means when the cache is not available, username and password will be prompted using a CallbackHandler.

Attention: Unfortunately, there's no way to inform Java that LSA should be looked up ahead of the file cache. So, you should always make sure that the %HOME%\krb5cc_userid file does not exist when you want to use the LSA. This file is generated by the kinit.exe command, so don't run it if you wish to use the credentials from the LSA cache.

On the other hand, if you want your program working even if the LSA cache is not available, normally you choose one of the following:

1. Run kinit.exe (comes with Java) before running the Java app, this will create a credentials cache file %HOME%\krb5cc_userid, which goes the 1st way. The JAAS login config file is the same as the typical style. If you create the credentials cache file into a different pathname, specify the location using ticketCache="c:/path/to/file" inside the JAAS login config file.
2. Feed username and password to the Java program directly using a CallbackHandler, which goes the 4th way. Please specify doNotPrompt=false in the JAAS login config file. You can provide an instance of CallbackHandler at the creation time of LoginContext if the JAAS call style is used (see the next section), or Java will create a new instance of the type specified by the security property auth.login.defaultCallbackHandler. For direct JGSS without JAAS, if this security property is not given, the internal text-based callback handler will be used.

JGSS calls

There are 2 ways to start JGSS:

1. Use JAAS to generate a Subject that contains the initial credentials, and call JGSS from this subject:

   ```
   LoginContext lc = new LoginContext(name, callback);
   lc.login(); lc.commit();
   Subject.doAs(lc.getSubject(), /* JGSS-API calls... */)
   ```

   In this case, you can choose whatever login entry name in the JAAS login config file. Read more for details.
2. Direct JGSS:

   ```
   /* JGSS-API calls... */
   ```

   In this case, the JAAS config file's entry name MUST be the standard entry name (com.sun.security.jgss.krb5.initiate), and you must set -Djavax.security.auth.useSubjectCredsOnly=false on the Java command line. Read here for details.

Other APIs that use JGSS

In Java, there are 2 other APIs that call JGSS-API internally.

1. SASL using JGSS as the mechanism. Read here and here for details.
2. HTTP/SPNEGO. Read here for details. Please note that the system property javax.security.auth.useSubjectCredsOnly is now default false for HTTP/SPNEGO now.

Please note that when initial credentials are not available from the cache (neither from a file nor the LSA), HTTP/SPNEGO behaves different in username and password providing. Instead of the JAAS callback model, java.net.Authenticator is used. Read the doc mentioned above.

Programming the server side

Service name

The biggest difference here from the client side is that there's no such concept as a native keytab, which means a JGSS server program cannot simply "RunAs" a Windows service account and uses the encryption key for that account. To make server side JGSS programming on Windows available, a special step is needed to create a mapping service name and a keytab file, by using the Microsoft provided tool ktpass.exe.

For example, if the AD domain name is AD.LOCAL, and you'd like to run a service called myservice on the host machine.ad.local, you can perform these steps on your AD server:

1. Create a normal user account (say myservicemachine) inside AD.LOCAL, any password is OK.
2. Call "ktpass -princ myservice/machine.ad.local@AD.LOCAL -mapuser myservicemachine@AD.LOCAL -out x.keytab +rndPass" to create a SPN mapping to the user account, and generate a keytab file x.keytab. The password is regenerated with a random value so the password you give in step 1 is useless.

Now, put the x.keytab file into a secret place that only your service application can read. The server side JAAS login config file would look like:

```
com.sun.security.jgss.krb5.accept {
    com.sun.security.auth.module.Krb5LoginModule
    required
    storeKey=true
    useKeyTab=true
    keyTab="c:/secret/path/to/x.keytab"
    principal="service/machine.ad.local"
    isInitiator=false;
};
```

Here you need to provide the location of the keytab file. Otherwise, Java will try to locate this file in these locations (ordered by):

1. default_keytab_name in the [libdefaults] section of krb5.ini, or
2. %HOME%/krb5.keytab

Note: isInitiator=false is specified here so that the application acts as a pure server side program that will never try to authenticate itself to the KDC. This is useful when it cannot communicate directly with the KDC.

JGSS calls

Just like the client side, you can use JAAS to create a Subject and call JGSS-APIs through this subject, or calls JGSS methods directly. In the latter case, please specify

```
   -Djavax.security.auth.useSubjectCredsOnly=false.
```

Delegations

To enable delegations, both configurations and programming on needed.

Configuration at client side (the delegated)

In order for the credentials of the client to be delegatable to a service, if the initial TGT is acquired the Java way, please add forwardable=true into the [libdefaults] section of krb5.ini. If from LSA, make sure the "Account is sensitive and cannot be delegated" is NOT set in AD account settings.

Configuration at server side (the delegator)

In order to use the delegated credentials from the client, we suggest the service needs to be configured to be allowed receiving delegations. For a computer account, find the delegation tab, or for a user account, find the account tab, check "Trusted for delegation". The turns on the OK-AS-DELEGATE for the service ticket. A Windows native client program needs this flag to enable delegation. A Java client MAY respect this flag later (currenrly NO except for HTTP/SPNEGO).

Programming

Please call GSSContext.requestCredDeleg(true) on the client side. Read here for details.

Trusts between Domains

If you have already setup cross realms trusts in multiple AD domains, please add the [domain_realm] section into the client side's krb5.conf file so that Java can correctly locate the realm for a requested service. Like this:

```
[domain_realm]
.this.com = THIS.COM
.that.com = THAT.COM
```

With this configuration, when a client on THIS.COM tries to connect to a service service/host.that.com, Java can correctly figured out that the service belongs to another realm THAT.COM and perform proper inter-realm authentications.

Other Windows Platforms

This article talks about Kerberos programming on the Windows platform. The typical KDC is Windows Server 2003. The typical client is Windows XP, and the typical server is Windows Server 2003 or Windows XP. There're some minor issues for other flavors of Windows versions.

Windows 2000 Server

Note that before SP4 of Windows 2000, there's no need to specify the allowtgtsessionkey registry key.

Windows Server 2008

In Windows 2008, the AES etype is supported. In order to use AES256 as the encryption etype, please download and enable Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files. Read the "Other downloads" section for details.

In Windows 2008, you cannot request a service ticket for a normal user, since Windows only allows user2user communications with a normal user using a special Microsoft defined Kerberos extension. In fact, Windows KDC simply does not issue a service ticket targeting a normal user.

There's a workaround to allow normal service ticket for a normal user on Windows 2008. Call setspn -a service/host username, a SPN will be created for this user. After this step, the client can acquire a normal service ticket targeting either the username or the SPN. This will also lure out the delegation tab for a user so that you can allows delegation on the user.

Debugging

These options are most useful in debugging a JGSS program:

1. Add -Dsun.security.krb5.debug=true on Java command line
2. Add debug=true into JAAS login config file
3. Inspect networking packets using a sniffer

Also, remember to always use the latest version of JRE/JDK. Some bugs may have already been fixed. The new versions may also show better debug information.

Frequently Asked Questions (FAQ)

Note: Please remember to always read the JGSS troubleshooting guide first. Some of the following case are included in that guide, and some are Windows-specific.

- JGSS complains that my JAAS config file has errors: Make sure it has correct format, the semi-colons are always there. Sometimes you need to put "" around a file name or principal name
- I have a keytab, but JGSS still asks me for password: Make sure the keytab path is correct. Maybe you need to provide a full path, maybe you need to add "" around it. Also, use "/" or "\\" as path separator.
- The client seems not login as an AD account: Sometimes you forgot to login as an AD account, or, if there are any network problems and your Windows client automatically goes to offline login mode. Use the Microsoft klist.exe or kerbtray.exe to see if a TGT is available in the LSA.
- The debug output shows native TGT is loaded, but does not use it: This happens when the session key inside TGT is not readable. For clients before Vista, use kerbtray.exe to see if encryption type for session key is null. For Vista, look at the debug output to see if the key are all zero. If so, please setup the allowtgtsessionkey registry key.
- Kerberos is never called (no Kerberos debug info after I add -Dsun.security.krb5.debug=true): Make sure various configurations are in place, which includes JAAS login conf file, krb5.ini file (or kdc/realm system properties). Also, if you don't use JAAS explicitly, make sure javax.security.auth.useSubjectCredsOnly is set to false.
- Delegation in HTTP/SPNEGO fails: Make sure in Windows AD computer settings, the allow delegation box is checked.
- Credentials not available: You don't have a native cache. Try to re-login as an AD account, or, consider the LSA-less way, say, -kinit.exe= or callback.
- Checksum failed (or other encryption/decryption errors): If you are using username/password callbacks, possibly the password is wrong. If you are using a keytab (on the server side), possibly the keytab contains a bad key.
- The Kerberos principal is not mine: Make sure there's no krb5cc_userid file inside your home directory. Java always uses this credential cache even if the user is logged in an AD user. Remove it.
- EType not supported: Latest Java (update releases of all versions) already supports RC4-HMAC, which is the default etype used on Windows. You needn't specify default etypes for either the session key or ticket in krb5.ini.
- Cannot find server name in Kerberos database: Have you mapped the service principal name(SPN) correctly? Please note that an SPN cannot be mapped to multiple accounts. Also, you must use the full qualified domain name in the ktpass command.
- Server name (as seen in the debug output) is not FQDN (full qualified domain name), or becomes simply numeric IP address: Make sure DNS is correctly configured. JGSS uses InetAddress.getCanonicalHostName() to get the FQDN of the server's hostname. Write a tiny program to check it.
- Cannot access Windows services like IIS: Make sure "Do not require pre-authentication" is not checked in AD user setting. Windows native services needs pre-authentication to provide PAC info in tickets.
- Cross realm failed: Read into debug outputs, especially the TGS-REQ info. Make sure the service principal name is correct. If the Windows DNS server is not configured correctly, it may not return the correct full qualified host name.
- Invalid option setting in ticket request: There are some options inside the [libdefaults] section of krb5.ini (say,

forwardable=true, proxiable=true etc) which are used to provide KDCOptions when requesting a TGT using the AS-REQ message. In this case, when a TGT is returned, its TicketFlags are compared to the options here. Since in this paper we're mainly talking about TGT from the Windows LSA cache, these options are useless. Proving too many of them will only bring conflicts with your native TGT.

Known Issues

If an AD account is also added into local administrator group on the client PC, Microsoft restricts such client from getting the session key for tickets (even if you set the allowtgtsessionkey registry key to 1). The workaround is: Just forget you're a logged in user, call kinit.exe. Do not depends on LSA credential cache.

In a recent hotfix (should be included in Vista SP1), this restriction is lifted for normal service tickets. However, it still applies to TGT. Since Java uses TGT to acquire tickets for other services (the standard Kerberos process), this update provides no benefit to JGSS programming on Windows. Furthermore, even if the implementation of Java is changed to read service tickets from the LSA cache, it still cannot perform delegation, since a TGT is always needed in that case.

Useful Tools

1. KERBTRAY.EXE, KLIST.EXE. SETSPN.EXE, KTPASS.EXE from Microsoft.
2. Any network packet sniffer. For example, Windows Netmon, Wireshark.
3. Web browsers, with HTTP header viewer (for example, LiveHTTPHeaders for Firefox, and iehttpheaders for IE) are useful in debugging HTTP/SPNEGO programs.

References

1. RFC 4120, 4121, 3961, 3962
2. MSDN doc on Kerberos and Active Directory