Fermilab

Kerberos Module for Java

Reference Guide

Andrey Petrov, apetrov@fnal.gov
Version 3.1.0 • October 12, 2005

This document describes the reusable software library providing a uniform and simplified procedure of strong user authentication in Java, based on Kerberos V5 protocol. The existing JRE implementation is extended to offer a better interoperability with various user environments, memory credential cache support, and additional functions. The ultimate goal of this project is to make the authentication procedure inconspicuous for the users and straightforward for the developers.

Most of the information in this Guide is based on the previous draft paper [1]. The API and implementation details, however, have been significantly changed since that time. Thus, a programmatic version described in the draft is obsolete and should not be used anymore.

Table of Contents

1. Introduction....1
2. Basic Concepts....2
3. Kerberos Module Overview....4
4. How the Authentication Works....5
5. Credentials Sharing....7
6. Configuration....8
7. Testing....10
8. Authentication in Web Browsers....13
9. Perspectives....17

1. Introduction

Continual expansion of on-line threads in the recent years forced the software developers to think of making improvements in their security tools. Most mechanisms of communication channel protection and access control depend on a proper authentication of principals, i.e., users and services. Once logged in, a principal must be able to present its credential everywhere throughout the system. This authentication procedure should not be too complicated, because otherwise the users will pay no heed to it.

It is hard to create a reliable and fair authentication system from scratch. For the application

software developers, taking advantage of an external security infrastructure is a more favorable way, especially if such a system is already in use. This approach provides yet another benefit: Utilization of a trusted third-party system increases confidence of the users and those who enforce security policies.

A while ago Fermilab adopted Kerberos V5 protocol as a uniform security solution. That infrastructure, including client utilities, has been installed on a vast number of computers throughout the lab. When people started using it for common network services, such as *rlogin* and *ftp*, it becomes a routine to get the fresh Kerberos ticket every morning. In this way, almost all the users have had a reliable and platform-independent digital proof of identity on their desktops.

It looked very attractive to use this kind of authentication in Java applications, as well. The Java Runtime Environment (JRE) included an implementation of Kerberos and GSS-API, thus no additional libraries were needed. That would make the logging procedure swift and less annoying than asking a password: As soon as a valid credential is available on the desktop, applications could use it to verify the user's identity and establish secure communication channels.

In reality the JRE's implementation of the Kerberos protocol has several drawbacks, which make it hard to use without changes in the existing environment.

1. In order to obtain credentials, the `com.sun.security.auth.module.Krb5LoginModule` class uses an inconsequential searching procedure. As a matter of fact, each operating system has a number of specific locations where the credentials can be found. Whether a Kerberos ticket is good, can be decided only after it has been read and parsed. A `LoginException` should be thrown only if no credentials have been found at all, or all of them were bad. It is acceptable to find an invalid ticket, if the subsequent readings return a valid one.
2. Memory credential caches not implemented (with the exception of Windows LSA).
3. The configuration is different when the protocol is used from web-startable applications, applets, or as a server-side component. Some files have to be locally pre-installed.
4. The GSS-API is far too general and incomprehensible for the occasional application developers [2].

The described Kerberos Module for Java (KMJ) addresses these and few other issues by extending the JRE and adding some new functionality. The product can be used as a base authentication component for various Java application, on both server and client sides.

2. Basic Concepts

This chapter gives some basic description of Kerberos protocol and GSS-API for the purpose of this guide.

Kerberos is a network authentication protocol defined in RFC 4120 [3] as:

> "... a means of verifying the identities of principals, (e.g., a workstation user or a network server) on an open (unprotected) network. This is accomplished without

> relying on authentication by the host operating system, without basing trust on host addresses, without requiring physical security of all the hosts on the network, and under the assumption that packets traveling along the network can be read, modified, and inserted at will."

Some explanation of the protocol can also be found in [4]. Here is an outline of how it works:

- Kerberos authenticates *principals*. The principal is either a user or a service.
- Each principal is required to have its *key* registered with the *authentication server* (AS). The key is either a value derived from the user's password, or a random value stored in a keytab file.
- In order to get initially authenticated, the client uses an authentication program, such as *kinit*, to exchanges some data with AS. The client principal's key is used as an encryption key in this exchange. If the authentication succeeds, the clients obtains a *ticket-granting ticket* (TGT).
- The TGT usually remains valid for several hours and is stored in a credential cache. The actual cache type (file or memory), its location, and the data format may vary among different authentication programs and operating systems.
- Every time when the client is going to contact some remote service, it requests a ticket from the *ticket granting server* (TGS) for that service principal, using the existing TGT.
- The local credentials can be forwarded to a remote host by means of GSS-API.

The **Generic Security Service Application Program Interface** (GSS-API) [5] was designed to describe how various mechanisms should be used to transfer credentials between two hosts. RFC 4121 [6] specifies the way GSS is applied to Kerberos.

GSS aims to create a *common security context* for two peers. In order to establish the context, the two nodes need to perform an authentication by exchanging one or several tokens. The authentication can be either mutual, when both nodes present credentials to each other, or non-mutual, when only the client is authenticated to the server.

A GSS context can be used to learn the names of both parties, and to apply some per-message security services on a communication channel between them. These services are: encryption, message integrity codes (MICs), sequence detection, and replay detection.

Sun included an **implementation** of Kerberos and GSS-API in JRE starting from the version 1.4. Technical notes can be found in [7].

Kerberos authentication is a part of the Java Authentication and Authorization (JAAS) service. The principal top-level class, responsible for obtaining credentials on the local computer, is `com.sun.security.auth.module.Krb5LoginModule`. Its weaknesses were discussed earlier.

The principal class of the GSS-API implementation is `org.ietf.jgss.GSSContext`. Kerberos mechanism is used with a proper object identifier (OID).

3. Kerberos Module Overview

Kerberos Module for Java supports the following functions:

- An automated credential discovery on multiple operating systems;
- Reading of memory credential caches;
- The keytab authentication of service principals without TGT requests;
- A simplified and uniform configuration procedure for applets, web-startable applications, and server-side components;
- A simplified API for sharing credentials between two peers;
- The web authentication framework;
- Integration with custom user and logging databases through pluggable modules.

All the binaries (compiled for VM 1.4) and configuration files are shipped in a single jar file, `kerberos-client.jar`. The module will run on JRE 1.4.2 and 1.5. No additional libraries are required (with an exception of the Tomcat authentication components that are supposed to be used inside the web server environment). The Kerberos configuration file `krb5.conf` included in the public distribution may need to be customized as described in the following chapters. After that, the project jar should be rebuilt and sealed with a new signature.

The project's root package is `gov.fnal.controls.kerberos`. It contains two principal classes:

`KerberosLoginContext`

: The main class performing the authentication of a local principal and sharing the credential with a remote host. The class is a singleton without public constructors. Its shared instance can be obtained from the static `getInstance` method. Can also be started from the command line as described in § 7.

`KerberosLoginApplet`

: The client-side applet with a graphical user interface for the web authentication.

Other packages contain various classes which are, in most cases, not used directly:

`catalina`

: Components for the Tomcat web server.

`login`

: An implementation of the credential reading procedures, including native libraries for the memory cache access.

`util`

: Miscellaneous useful gadgets.

4. How the Authentication Works

The main class, `KerberosLoginContext`, is an extension of `javax.secutity.auth.login.LoginContext` that defines most the general login and logout procedures. In case of the Kerberos authentication, a *login* simply means reading of the current principal's credential, which is either a cached ticket-granting ticket, or a secret key stored in the keytab. The credential must already exist on the local system, because the Module does not request ticket-granting tickets from the server. The main reason for this is not to deal with the Kerberos passwords, which are highly sensitive information. It is quite difficult to figure out whether a program is running on X terminal where password exposure is unwanted.

A particular way how the credential is initially obtained depends on the system environment and the adopted Kerberos configuration. The Module was tested on the data formats provided by several client utilities:

Table 1. Supported Kerberos Clients.

Operating system	Client	Typical Credential Location
Tickets caches:		
Windows 2000/XP	*kinit*, *Leash32* (older versions)	• `%KRB5CCNAME%` • `{java.io.tmpdir}/krb5cc` • `{user.home}/krb5cc_{user.name}`
	kinit, *Leash32* (newer versions)	operating system memory
	WRQ Reflection	• `{user.home}/My Documents/Reflection/{user.name}.cch`
Linux SunOS FreeBSD	*kinit*	• `$KRB5CCNAME` • `/tmp/krb5cc_{uid}` • `{user.home}/krb5cc_{user.name}`
Mac OS X	default client	operating system memory
Keytabs:		
all	*kadmin*	• `{user.home}/krb5.keytab`

A concrete search order is specified in the `login.conf` file. The parameters can be adjusted to comply with the existing environment.

Reading tickets from the memory caches is accomplished through the Java Native Interface (JNI). The project's jar file contains individual native libraries for each supported operating system. On Windows, `krbcc32.dll` (included in *Leash32*) is required. On Mac OS X all needed system libraries come up pre-installed. When used inside an applet, the native libraries must be properly unloaded from the virtual machine. Otherwise, the applet's class loader architecture would not allow the authentication on several web sites in a single browser, throwing a “Library Is Already Loaded” exception. The Module fixes this with an additional isolation level between class loaders. It assures that the unused native libraries will be properly collected along with the corresponding classes.

Note: Kerberos Module for Java was designed to work only with the MIT version of

Kerberos. The Microsoft has its own implementation, included in Windows 2000 and XP, where the credentials are stored in the memory and accessed through the Local Security Authority (LSA) API. Kerberos Module does not support Windows LSA! Generic Java classes are seemed to be able to use LSA, though.

Like in the parent `LoginContext`, the authentication procedure in `KerberosLoginContext` is governed by two methods:

`login`

Reads the principal's credential. In case of a failure, throws a `LoginException`, otherwise quits quietly.

`logout`

Erases an authentication data that can be stored inside the module.

The `getSubject` and `getPrincipal` methods can be used to learn who the current principal is.

Technically, the principal authentication procedure is a loop with many ifs: The Module examines all the locations (plain files and memory adapters) specified in the login configuration file and tries to open them one after another. When some data is seemed to be available, reading is attempted. If the data format matches an expectation, the ticket (or secret key) is parsed and analyzed. For a ticket, this includes checking up the server principal name, flags, and the expiration date. If all the trials succeed, the search routine is terminated and the found credential is returned. Otherwise, if no valid credential is found, a `LoginException` is thrown.

We found it to be particularly important to provide precise diagnostics in case of login failures. The authentication procedures is often confusing for the users, as they do not understand exact requirements. The system must distinguish few situations, such as: nothing that looks like a ticket can be found at all, the ticket format is incorrect, the ticket has expired, invalid ticket, and so forth. The Module implements this facility by weighting each partial reading error, and returning only the most significant one if everything has failed. For example, if the user has an expired ticket in one file and some garbage in the second one, the final reason is going to be “Your Ticket Has Expired”, not “Bad Cache Format”, notwithstanding the order they were found.

In a typical situation, the user principal is authenticated with a cached ticket-granting ticket. The service principal, in its turn, is authenticated through a keytab. If the service principal is required only to establish a GSS context (see § 5), there is no need to obtain a TGT. Unlike the JRE implementation, the Module does not request it in this situation. Client-type principals occasionally can also use keytabs. For example, it makes sense for public consoles, on which the users do not have individual accounts. Assuming the risk it may cause, such capabilities are present anyhow in real life and must be covered by the same authentication technique.

It is important to realize that the `KerberosLoginContext`'s *login* operation does not actually authenticate the user. The acquired piece of information only *looks* like a credential. In fact, the credential storage could easily be tampered with, the ticket could be revoked, or there may be other reasons unknown for the client, which make the ticket or secret key invalid. As stated in

[3], to validate a credential, one must use it on the server it is intended for. In our case, the ticket-granting ticket must be presented to an appropriate TGS, and the secret key must be used either on the authentication server to get a TGT, or to accept a GSS context.

5. Credentials Sharing

After two peers get authenticated, they can present their identities to each other and establish a secure communication channel. The GSS-API specification [6] determines augment protocols and notions for the developers. According to this concept, two peers possessing their Kerberos credentials can exchange some information between each other and establish a *common security context*. Inside the context a credential delegation and several per-message security services are available. These services are: encryption, message identity codes (MICs), sequence detection, and replay detection. They are applied to the data by wrapping and unwrapping data streams.

In Java, this common security context is implemented in `java.ietf.jgss.GSSContext` class. Detailed instructions on its establishing and usage are given in [7]. In practice, however, the given abstract concept is poorly understandable by many occasional programmers.

The `KerberosLoginModule` implements a few methods for GSS context establishing that will fit most of the user applications. These methods are partially integrated with the Kerberos authentication routines and use common configuration namespace. Many technical details are hidden from the end users (for example, conversions between Kerberos principals and GSS names).

The GSS context is always initiated on the client side by one of the following methods:

`initSecContext(InputStream, OutputStream, ServiceName, byte)`

Requests the security services as specified by the byte argument. Input and output streams have to be connected with the server in some application-specific way (for example, via a TCP/IP socket). Depending on a particular set of features, the parties can be mutually authenticated, or only the client authenticates to the server. It takes 1 (one-way mode) to 3 (mutual mode) transactions via the streams to complete the request. The method returns a `GSSContext` instance that can be used further to get remote server name, wrap data streams, etc.

`initSecContext(ServiceName)`

A trivial method. Performs only a one-way authentication of the client on the server (this can be used, for example, in a web browser). Unlike in the mutual type, does not establish per-message security. The method returns a BASE64-encoded security token ought to be sent to the server.

On the client side, two similar methods are employed. Both of them return a `GSSContext` instance that keeps the client's identity and can be used to apply per-message security in the mutual mode:

`acceptSecContext(InputStream, OutputStream)`

The input and output streams have be connected to the output and input streams on the

client side, correspondingly.

`acceptSecContext(String)`

A trivial method. Accepts a BASE64-encoded token from its client counterpart.

In order to initiate and accept a security context, two peers must know the name of a GSS service this context is associated with. RFC 1964, § 2.1.2 defines the *host-based service name* format as:

```
service@hostname
```

The GSS service name is derived from the corresponding name of a Kerberos's service principal acting on the server side, as shown below. To avoid confusion in distinguishing of two name formats, a new `ServiceName` was introduced.

`host/foo.fnal.gov@FNAL.GOV` *(Kerberos)* ➔ `host@foo.fnal.gov` *(GSS)*

Special characters, such as "/" and "@", in the GSS name's *service* part must be escaped with a backslash. Inside `ServiceName`, this is done automatically:

`daeset/bd/foo.fnal.gov@FNAL.GOV ➔ daeset\/bd@foo.fnal.gov`

6. Configuration

As the Kerberos protocol in Java must work in the existing global environment, some initial configuration is always required. The Module does not rely on any configuration files or environment variables that can be found on an individual computer (except those environment variables used to find cached credentials). To be self-sufficient, though, the module's jar file must already include all the configuration data. Some tuning can be done through Java properties. All used properties must be set at the program startup, before the `KerberosLoginContext` class is statically initialized.

6.1. Files

The `kerberos-client.jar` contains two configuration files:

`/META-INF/krb5.conf`

The main Kerberos configuration. Although a similar file usually exists on all the computers, various versions of Kerberos clients can use different places to store it. So, there is a chance to find an obsolete data in `/etc/krb5.conf` instead of the working configuration. By default, the Module copies this file from the jar to a temporary local directory and then uses that copy. In order to override this behavior and read a local configuration file instead, the name of such a file should be specified in `java.security.krb5.conf` property.

As an alternative to the configuration files, the two Java properties can be set: `java.security.krb5.realm` and `java.security.krb5.kdc`

Note: The public distribution contains some sample data in `/META-INF/krb5.conf`. If `java.security.krb5.conf` (or `...realm` and `...kdc`) are not specified, you must update

that file with your proprietary configuration and rebuild the jar with the provided Ant script.

`/META-INF/login.conf`

Defines the credential search order for various operating systems. Update this file if in your existing configuration the cached tickets and keytabs are stored elsewhere.

6.2. Java Properties

`gov.fnal.controls.kerberos.keytab`

The keytab file name. Should be specified if the principal's credential is stored in a keytab file different from `{user.home}/krb5.keytab`.

`gov.fnal.controls.kerberos.principal`

Default principal name. Required only if the authentication is conducted through a keytab file. If not specified, the expected keytab principal is reconstructed from the current user name and the default realm (this assumption will not work for service principals).

If there is a credential cache, this property is not required. But if it is set, the value is used as an additional filter for the tickets.

`java.security.krb5.conf`

Name of the Kerberos configuration file. Set this property only if the default `krb5.conf` does not exist in `kerberos-client.jar`, or should not be used.

If the `...conf` property is not set, the `KerberosLoginContext` tries to use a local copy of `/META-INF/krb5.conf` (preferred way). However, if both `...realm` and `...kdc` properties are set, they take precedence.

`java.security.krb5.realm`

`java.security.krb5.kdc`

Kerberos default realm and the key distribution center address. Can be used if no `krb5.conf` is available. These properties must always come together.

`gov.fnal.controls.kerberos.default_realm`

An internal property which keeps the default realm name. It it set automatically and should not be changed from applications.

`java.security.auth.login.config`

Login configuration file URL. It is set automatically during the Module start up and points to the `login.conf` inside the project's jar file. Typically, should not be changed.

6.3. Configuration Checklist

As a matter of fact, the Kerberos Module configuration requires to answer three questions:

1. Which Kerberos configuration file to use?

(a) **/META-INF/krb5.conf inside kerberos-client.jar.**

- ✔ Replace the sample krb5.conf with your proprietary configuration and rebuild the archive.

(b) **Some local configuration file, e.g. /etc/krb5.conf.**

- ✔ Set the file name in java.security.krb5.conf.

(c) No configuration files at all.

- ✔ Set both java.security.krb5.realm and java.security.krb5.kdc properties.

2. What kind of credentials is used?

(a) **A ticket-granting ticket (most of the user principals).**

- ✔ Do nothing.

(b) **A secret key (service principals and special user principals).**

- ✔ If the principal name is not {user.name}@{...default_realm}, put the right value in the gov.fnal.controls.kerberos.principal property.
- ✔ If the keytab name is not {user.home}/krb5.keytab, put the right value in the gov.fnal.controls.kerberos.keytab property.

3. Can the Module automatically find credentials?

(a) **Yes.**

- ✔ Do nothing.

(b) **No.**

- ✔ Upgrade the default /META-INF/login.conf and rebuild the archive.

7. Testing

To test out the Module in a particular configuration, the KerberosLoginContext class can be launched from the command line as a standalone program. The common call syntax is:

```
java -cp kerberos-client.jar ↓
  gov.fnal.controls.kerberos.KerberosLoginContext options
```

Instead of java, you may have to use some more precise syntax for pointing out the required virtual machine, such as $JAVA_HOME/bin/java. Also make sure that the kerberos-client.jar is available and contains all needed configuration files. To set a system property, use -D*name*=*value* construct.

The program can be used in three modes. For a brief help, use -h option.

7.1. Authentication Mode

Requires no options. The current principal is authenticated by reading its credential from a ticket

cache or a keytab file:

```
bash-2.05$ java -cp kerberos-client.jar gov.fnal.controls.kerberos.KerberosLoginContext
Utility for testing Kerberos authentication through GSS over TCP/IP.
    2005 (c) Universities Research Association, Inc.
    Author: Andrey Petrov, apetrov@fnal.gov.
    Version: 1.7
Settings:
    java.security.auth.login.config=jar:file:/export/users/apetrov/kerberos-client.jar!/MET
A-INF/login.conf
    java.security.krb5.conf=/var/tmp/krb5.conf
    java.security.krb5.realm=null
    java.security.krb5.kdc=null
    gov.fnal.controls.kerberos.principal=null
    gov.fnal.controls.kerberos.keytab=_void_
    gov.fnal.controls.kerberos.default_realm=FNAL.GOV
Starting authentication...
Oct 4, 2005 10:11:31 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Reading cache FILE:/tmp/krb5cc_p3644
Oct 4, 2005 10:11:31 AM gov.fnal.controls.kerberos.KerberosLoginContext login
INFO: Authenticated as apetrov@FNAL.GOV until Wed Oct 05 10:50:31 CDT 2005 (1125ms)
Finished
```

As can be seen on the screen shot, the program prints out values of all relevant system properties, either set by the user, or defined automatically.

The following example shows the service principal authentication (mind additional properties):

```
bash-2.05$ java -cp kerberos-client.jar -Dgov.fnal.controls.kerberos.keytab=[redacted]/
[redacted]/daeset_keytab -Dgov.fnal.controls.kerberos.principal=daeset/bd/dpe08.fnal.gov@FNAL.G
OV gov.fnal.controls.kerberos.KerberosLoginContext
Utility for testing Kerberos authentication through GSS over TCP/IP.
    2005 (c) Universities Research Association, Inc.
    Author: Andrey Petrov, apetrov@fnal.gov.
    Version: 1.7
Settings:
    java.security.auth.login.config=jar:file:/export/users/apetrov/kerberos-client.jar!/MET
A-INF/login.conf
    java.security.krb5.conf=/var/tmp/krb5.conf
    java.security.krb5.realm=null
    java.security.krb5.kdc=null
    gov.fnal.controls.kerberos.principal=daeset/bd/dpe08.fnal.gov@FNAL.GOV
    gov.fnal.controls.kerberos.keytab=[redacted]/daeset_keytab
    gov.fnal.controls.kerberos.default_realm=FNAL.GOV
Starting authentication...
Oct 4, 2005 10:38:30 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Reading cache FILE:/tmp/krb5cc_p3644
Oct 4, 2005 10:38:30 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Skipping cache FILE:/tmp/krb5cc_3842
Oct 4, 2005 10:38:30 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Skipping cache FILE:/home/apetrov/krb5cc_apetrov
Oct 4, 2005 10:38:30 AM gov.fnal.controls.kerberos.login.KeytabReader login
INFO: Reading keytab FILE:[redacted]/daeset_keytab
Oct 4, 2005 10:38:30 AM gov.fnal.controls.kerberos.KerberosLoginContext login
INFO: Authenticated as daeset/bd/dpe08.fnal.gov@FNAL.GOV until null (312ms)
Finished
```

7.2. Server Mode

The program authenticates the local service principal, and than opens a TCP/IP port to listen for incoming client requests. The syntax is:

```
KerbrosLoginContext server [port] ,
```

where *port* is an optional of the TCP/IP port which should be opened. The default is 4747.

Starting the utility:

```
bash-2.05$ java -cp kerberos-client.jar -Dgov.fnal.controls.kerberos.keytab=[redacted]/
[redacted]/daeset_keytab -Dgov.fnal.controls.kerberos.principal=daeset/bd/dpe08.fnal.gov@FNAL.G
OV gov.fnal.controls.kerberos.KerberosLoginContext server
Utility for testing Kerberos authentication through GSS over TCP/IP.
    2005 (c) Universities Research Association, Inc.
    Author: Andrey Petrov, apetrov@fnal.gov.
    Version: 1.7
Settings:
    java.security.auth.login.config=jar:file:/export/users/apetrov/kerberos-client.jar!/MET
A-INF/login.conf
    java.security.krb5.conf=/var/tmp/krb5.conf
    java.security.krb5.realm=null
    java.security.krb5.kdc=null
    gov.fnal.controls.kerberos.principal=daeset/bd/dpe08.fnal.gov@FNAL.GOV
    gov.fnal.controls.kerberos.keytab=[redacted]/daeset_keytab
    gov.fnal.controls.kerberos.default_realm=FNAL.GOV
Starting authentication...
Oct 4, 2005 11:15:07 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Reading cache FILE:/tmp/krb5cc_p3644
Oct 4, 2005 11:15:07 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Skipping cache FILE:/tmp/krb5cc_3842
Oct 4, 2005 11:15:07 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Skipping cache FILE:/home/apetrov/krb5cc_apetrov
Oct 4, 2005 11:15:07 AM gov.fnal.controls.kerberos.login.KeytabReader login
INFO: Reading keytab FILE:[redacted]/daeset_keytab
Oct 4, 2005 11:15:07 AM gov.fnal.controls.kerberos.KerberosLoginContext login
INFO: Authenticated as daeset/bd/dpe08.fnal.gov@FNAL.GOV until null (390ms)
Openning port 4747
Waiting for connection. Use Ctrl+C to interrupt.
```

Upon arriving of a request:

```
Connection requested from /131.225.[redacted]
Security context established (4232ms)
    established=true
    protReady=true
    srcName=apetrov@FNAL.GOV
    targName=daeset\/bd@dpe08.fnal.gov
    mutualAuthState=true
    integState=true
    confState=false
    replayDetState=true
    sequenceDetState=true
    lifetime=2147483647
Waiting for connection. Use Ctrl+C to interrupt.
```

7.3. Client Mode

In this mode, the local user principal is authenticated and then its credential is forwarded to the server through a TCP/IP connection. There must be another utility already running in the server mode.

KerbrosLoginContext client *service host*[:*port*] [*gss-opt*]*,

where

service

A “short” service identifier, used along with the destination address to create the full

host-based service name.

host

Destination server address.

port

Optional port on the destination server. The default is 4747.

gss-opt

GSS-API options (by default all of them are reset):

`+i`	request integrity check (message integrity codes),
`+c`	request confidentiality (encryption),
`+r`	request replay detection,
`+s`	request sequence detection.

```
bash-2.05$ java -cp kerberos-client.jar gov.fnal.controls.kerberos.KerberosLoginContext cli
ent daeset/bd dpe08.fnal.gov +i +r
Utility for testing Kerberos authentication through GSS over TCP/IP.
    2005 (c) Universities Research Association, Inc.
    Author: Andrey Petrov, apetrov@fnal.gov.
    Version: 1.9
Settings:
    java.security.auth.login.config=jar:file:/export/users/apetrov/kerberos-client.jar!/MET
A-INF/login.conf
    java.security.krb5.conf=/var/tmp/krb5.conf
    java.security.krb5.realm=null
    java.security.krb5.kdc=null
    gov.fnal.controls.kerberos.principal=null
    gov.fnal.controls.kerberos.keytab=_void_
    gov.fnal.controls.kerberos.default_realm=FNAL.GOV
Starting authentication...
Oct 4, 2005 11:26:34 AM gov.fnal.controls.kerberos.login.CacheLoginModule login
INFO: Reading cache FILE:/tmp/krb5cc_p3148
Oct 4, 2005 11:26:34 AM gov.fnal.controls.kerberos.KerberosLoginContext login
INFO: Authenticated as apetrov@FNAL.GOV until Wed Oct 05 10:50:31 CDT 2005 (231ms)
Connecting to dpe08.fnal.gov:4747...
Starting GSS transaction with daeset\/bd@dpe08.fnal.gov...
Security context established (4234ms)
    established=true
    protReady=true
    srcName=apetrov@FNAL.GOV
    targName=daeset\/bd@dpe08.fnal.gov
    mutualAuthState=true
    integState=true
    confState=false
    replayDetState=true
    sequenceDetState=true
    lifetime=2147483647
Finished
```

8. Web Authentication

With a growing number of web services, employing the Kerberos authentication for HTTP looks very attractive. However, at this time there is no general solution on how it should be done. The Module uses an approach that can fit many application. Here is a scenario:

1. The user obtains a local ticket-granting ticket, either with a Kerberos client, or by forwarding it from another host.
2. On the web page, an applet reads this ticket and requests establishing of a new security context The binary token created by GSS-API is sent to the web server in lieu of a password.
3. The server accept the token, creates a context and gets the user's name from it. Possession of the valid GSS context means a successful authentication. If the token is incorrect, forged, expired, etc., the context can not be established, and the user is rejected.

The implementation consists of a client-side applet, few web pages, and a server-side module. The client has been tested on all operating systems listed in Table 1 with various versions of Internet Explorer, Mozilla, Mozilla Firefox, and Safari web browsers. The browsers must have the Java Plugin ver. 1.4.2 or 1.5 already installed. The server side modules were developed for Tomcat server ver. 4.1, 5.0[1].

The procedure is using a *FORM* (not *BASIC*) type of authentication, in which the user name and password are entered in a special login page. The Module has the applet on this page that fills out the *password* field with a GSS token (the *name* field remains blank).

Here is how the *FORM* authentication works in Tomcat:

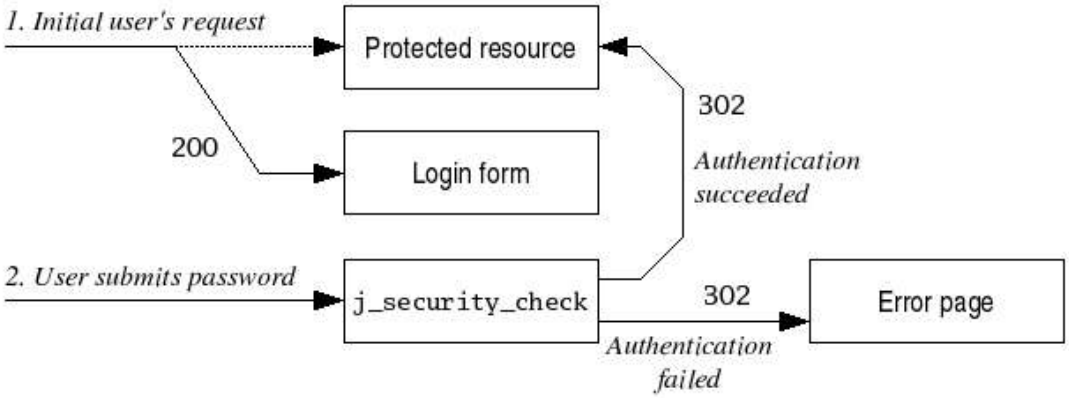

Fig 1. FORM Authentication in Tomcat

When an unauthenticated user tries to request a protected resource, the server silently (with the status code 200) redirects to the login page, containing a web form for the user name and password. This form submits data to a *j_security_check* path, pre-assigned to an authentication procedure. If the authentication succeeds, the user is redirected to the originally requested page (with the status code 302). If it fails, an error page is shown. Note, that if the original request had *POST* type, it is converted to *GET* upon the final redirection.

8.1. Client's Applet

The `KerberosLoginApplet` class implements a simple applet on top of `KerberosLoginModule`.

1 The current version won't compile and run with Tomcat 5.5, as the Catalina API has slightly changed. The differences are actually very small.

The login procedure begins immediately after the applet's `start` method is invoked. Its current status is shown in the GUI (the bottom line is a destination service name).

Firstly, the applet tries to read the local credential. This is done on the same way as described in § 4. If the valid ticket or the secret key is found, the applet initiates a new GSS context with the web server. The destination host name and the "short" service name must be specified in a `host` and `service` applet parameters, correspondingly.

As this procedure is used solely for the client authentication, there is no need to request any security services in `initSecContext`. The common security context can be established in one transaction, by sending a single bunch of data (a token) to the server. The applet encodes the token with the BASE64 algorithm and calls the `contextReady` JavaScript function on the parent web page, using the encoded value as a parameter. Than, the script sends the token back to the server.

8.2. Web Resources

The `login` subdirectory should be added to each web application wanted to use the authentication. Here is the directory contents:

`index.jsp`

A sample page used to test and debug the login procedure.

`krb5login.jsp`

The main login page that starts the applet and submits data to `j_security_check`. It also defines the current host name on the server side, and puts this value as a `host` parameter for the applet. A combination of `object` and `embed` tags is used as recommended in [8].

In the public distribution, the applet's archive `kerberos-client.jar` is loaded from the current directory. That can be changed to reflect an actual configuration.

`error.html`

Shows up if the authentication failed.

`warning.html`

Some custom login warnings and instructions. Referenced from `krb5login.jsp`.

`login.js`

`ok.html`

`login.js` is a simple JavaScript to start the authentication. In fact, it just opens `ok.html` in a separate window. The `ok.html` should be registered inside the protected area (in `web.xml`). Once reached, it closes the current browser window and reloads the parent one.

8.3. Server-Side Modules

On the Tomcat web server, the Kerberos authentication mechanism is provided by the `Krb5Realm` class (implementing the Catalina's `org.apache.catalina.Realm` interface). The class accepts a BASE64-encoded GSS token in lieu of a password, creates a GSS context, and, is succeeded, sets up the current user's principal. The security realm should be registered in the `server.xml` configuration file inside the `Host` container, as follows:

```
<Realm className="gov.fnal.controls.kerberos.catalina.Krb5Realm"
       servicePrincipal="service_principal_name"
       keyTab="keytab_name"
       userLookup="user_lookup"
       userLog="user_logger" /> ,
```

where:

service_principal_name

A name of the acting service principal. Will be set to `gov.fnal.controls.kerberos.principal` environment variable. The "`{0}`" construct will be automatically substituted with the current host name. For example,

```
daeset/bd/{0}@FNAL.GOV
```

on the host `dpe08.fnal.gov` will be converted to

```
daeset/bd/dpe08.fnal.gov@FNAL.GOV .
```

keytab_name

Full path to the keytab file

user_lookup (optional parameter)

The name of a class implementing the interface with a user database. By default, it is `gov.fnal.controls.kerberos.login.UserLookup`. Custom implementations must extend this class.

user_logger (optional parameter)

The name of a class implementing the interface with a custom security logging database. By default, it is `gov.fnal.controls.kerberos.login.UserLog`. Custom implementations must extend this class.

There may be a need to register few hosts that have to be authenticated according their addresses withoutKerberos. This is implemented by the special Tomcat valve:

```
<Valve className="gov.fnal.controls.kerberos.catalina.HostAuthenticator"
       address="address_mask"
       userLookup="user_lookup"
       userLog="user_logger" /> ,
```

where:

address_mask (optional parameter)

A regular expression used for rough address filtering to reduces the number of database calls. For example, the construct

```
^192\.168\.((2)|(22))\.\d{1,3}$
```

will allow only addresses `192.168.2.*` and `192.168.22.*` to go through.

`user_lookup`

The name of a class implementing the interface with a user database. The database provides a correspondence between host addresses and the user names. The default `gov.fnal.controls.kerberos.login.UserLookup` class must be extended, as it returns `null` for such all requests.

`user_logger` (optional parameter)

The same meaning as in the `Realm`.

We would suggest to use the Single Sign-On valve, too:

```
<Valve className="org.apache.catalina.authenticator.SingleSignOn" />
```

The specific protected area must be configured in the application's `web.xml` file, as described in the Tomcat User Guide.

9. Perspectives

The nearest plan is to implement a Transport Level Security protocol based on the Kerberos authentication. The mechanism described in RFC 2712 (and implemented in Java 1.5) looks controversial because it is always using the `host/MachineName@Realm` service name, generally assigned to other services, such as *telnet* and *ftp*.

References

[1] A. D. Petrov. Using Kerberos Authentication In Java. Draft Version. Beams-doc-1502. *http://beamdocs.fnal.gov/cgi-bin/public/DocDB/ShowDocument?docid=1502*

[2] A, Bosworth. ICSOC04 Talk. *http://www.adambosworth.net/archives/000031.html*

[3] C. Neuman et al. The Kerberos Network Authentication Service V5. Request for Comments (RFC) 4120, Internet Engineering Task Force, July 2005.

[4] B. Tung. The Moron's Guide to Kerberos. *http://www.isi.edu/gost/brian/security/kerberos.html*

[5] J. Linn. Generic Security Service Application Program Interface. Request for Comments (RFC) 1508, Internet Engineering Task Force, September 1993.

[6] L. Zhu et al. The Kerberos Version 5. Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2. Request for Comments (RFC) 4121, Internet Engineering Task Force, July 2005.

[7] M. Upadhyay and R. Marti. Single Sign-On Using Kerberos In Java. Sun Microsystems. *http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/single-signon.html*

[8] Using OBJECT, EMBED and APPLET Tags in Java Plug-in *http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/using_tags.html*