CoreUserDirectoryClient

v.M1-SNAPSHOT

CUD REST Interface Client User Guide

OUCS Systems Development and Support Team

2012-04-17

Table of Contents

1 Introduction

This document aims to describe how to use command-line client tools to interact with the Core User Directory (CUD) REST interface, for the extraction of data according to specific queries. These tools are intended to help with the automation of a batch-processing strategy.

Requests to the CUD REST interface are made using the HTTP GET method, with parameters included in the query string. Results are streamed in the HTTP response in a choice of formats: XML; JSON; CSV; Delimited Text with named delimiter and text qualifier.

A self service interface is also available to enable users to pre-build queries which can then be referenced by name in the query string. Access to the REST end points are over SSL/TLS, with the Kerberos GSS-Negotiate mechanism used for authentication.

Queries to the CUD REST interface use the Solr/Lucene query syntax. More information about Solr query syntax may be found at http://wiki.apache.org/solr/SolrQuerySyntax.

2 Parameters

As mentioned earlier, parameters are included in the query string. These should be URL-encoded (there are plenty of free URL-encoding utilities available both on and off the web).

Allowed parameters are as follows:

2.1 `q`

This is the parameter describing the Solr query, and is the only mandatory parameter.

Legal values are empty (no data will be returned), or a legal Solr query.

2.1.1 Example Request Parameters

An example of the Solr query "cud\:cas\:sso_username:unit1234", when encoded:

```
q=cud%5C:cas%5C:sso_username:unit1234
```

The Solr query "cud\:cas\:sso_username:unit1234 AND cud\:cas\:current_affiliation:oucs", when encoded:

```
q=cud%5C:cas%5C:sso_username:unit1234%20AND%20cud%5C:cas
%5C:current_affiliation:oucs
```

2.2 `fields`

The fields parameter allows the requestor to reduce the set of values returned for each record to only those attribute fields that are specified.

Legal values are a comma-separated list of attribute names.

2.2.1 Example Request Parameters

The fields parameter used to specify the Oxford email and CUD ID ("cud:cas:oxford_email,cud:cas:cudid"):

```
q=cud%5C:cas%5C:sso_username:oucs0098&fields=cud:cas:oxford_email
%2Ccud:cas:cudid
```

2.3 `format`

The format parameter in CUD queries is optional, and specifies the format in which the requested data will be returned. It may be omitted, in which case the default 'csv' format will be used.

Legal values are 'csv' (the default), 'xml', 'json', or 'text'. If 'text' is chosen you can optionally set the 'delimiter' and 'qualifier' parameters to further customise the output (See below).

2.3.1 Example Request Parameters

The format parameter being used to specify data returned as XML:

```
q=cud%5C:cas%5C:sso_username:unit1234&format=xml
```

2.4 `delimiter`

The delimiter parameter in CUD is optional, and only relevant when the 'text' format is used. For other formats this parameter will be ignored. The first character of the value set for this parameter will

be used as the field delimiter. If the text format is set, and this parameter is omitted, a comma will be used as the delimiter.

2.4.1 Example Request Parameters

The text format using a specific delimiter ("|"):

```
q=cud%5C:cas%5C:sso_username:unit1234&format=text&delimiter=%7C
```

2.5 `qualifier`

The qualifier parameter in CUD is optional, and only relevant when the 'text' format is used. For other formats this parameter will be ignored. The first character of the value set for this parameter will be used. The purpose of this parameter is generally to escape the field. If the text format is set, and this parameter is omitted, no additional qualifier will be used.

2.5.1 Example Request Parameters

The text format using a specific delimiter ("|") and qualifier ("@"):

```
q=cud%5C:cas%5C:sso_username:unit1234&format=text&delimiter=%7C&qualifier=@
```

2.6 `history`

The history parameter in CUD queries is optional, and indicates whether you wish the response to contain historical values for attributes and affiliations. You should only request this if it is required, and the default is not to include history.

Legal values are 'y', 'n' or omitted entirely.

2.6.1 Example Request Parameters

The history parameter used to request historical data in an XML format:

```
q=cud%5C:cas%5C:sso_username:unit1234&format=xml&history=y
```

3 Prerequisites

To automate access to the CUD REST interface a tool is required that will allow authentication using the GSS Negotiate protocol with a Kerberos service principal keytab.

The following tools offer this functionality and have been successfully tested to work with the CUD REST interface.

However, these tools are not the only way to access the CUD REST interface, there is no reason why programmatic solutions should not be developed if you have a requirement to.

3.1 curl

The curl tool may be used as long as it has been compiled with GSS-Negotiate and SSL support. Compiling with GSS-Negotiate support is dependent upon the availability of particular Kerberos libraries. The MacOS and linux operating systems are examples for which curl has been packaged with built-in GSS-Negotiate support, while curl built for the Windows OS may not have been.

To verify whether your version of curl has support for SSL and GSS-Negotiate, invoke curl with the '--version' option, as in the following example:

```
$ curl --version
curl 7.21.0 (x86_64-pc-linux-gnu) libcurl/7.21.0 OpenSSL/0.9.8o zlib/1.2.3.4 libidn/1.15 libssh2/1.2.6
Protocols: dict file ftp ftps http https imap imaps ldap ldaps pop3 pop3s rtsp scp sftp smtp smtps telnet tftp
Features: GSS-Negotiate IDN IPv6 Largefile NTLM SSL libz
```

If you see "GSS-Negotiate" and "SSL" in the third line of the output ("Features") you can use curl for this.

If the "GSS-Negotiate" (Negotiate authentication) feature is missing you can alternatively use the java-based command-line tool.

3.2 NegotiateRestClient.jar

This executable jar file includes a modified form of the Java SPNEGO library available from http://spnego.sourceforge.net. Modifications were necessary to allow its use with GSS-Negotiate (rather than almost-identical SPNEGO). The original library was licensed under the LGPL, and the modified version has inherited the same license. A copy of the LGPL is included within this Java archive, in the META-INF directory.

To use this tool you must have installed the prerequisite Java SE (Standard Edition) Runtime Environment (JRE). This is available as a free download from http://java.oracle.com . Alternatively an equivalent OpenJDK edition can be installed if it is available for your OS platform. The minimum required Java Runtime Environment (JRE) version is Java 7.

When the JRE is installed the tool can be invoked as follows:

```
$ java -jar NegotiateRestClient.jar
```

If a JRE is successfully installed, the output should include an error message, and a usage message resembling the following:

```
usage: java -jar NegotiateRestClient.jar [options]
 -c,--config <arg>         use specified login configuration file
 -D,--debug                enables extra debug output, use when verbose
                           error reporting is required
 -d,--data <arg>           sends the specified data in a POST request to
                           the HTTP server (required if -f/--data-file not
```

```
                          provided)
-f,--data-file <arg>      reads in data from a file and sends the data in
                          a POST request to the HTTP server (required if
                          -d/--data not provided)
-G,--get                  make all data specified with -d/--data to be
                          used in an HTTP GET request instead of the POST
                          request that otherwise would be used. The data
                          will be appended to the URL with a '?'
                          separator.
-h,--help                 display this message - all other options are
                          ignored if this option is specified
-m,--login-module <arg>   use specified login configuration module
                          section, this should be a labelled section in
                          the file specified with -c/--config
-u,--url <arg>            the target URL
```

4 Requesting Access

Before using the CUD REST interface you should have a Kerberos service principal in the OX.AC.UK realm, which should be authorised appropriately for access to CUD data sets.

4.1 Requesting a Service Principal

Email sysdev@oucs.ox.ac.uk to request a service principal for access to CUD, specifying the */itss* principal that should have administrative rights over that principal, and that the principal is intended to be used to access CUD data.

4.2 Requesting access to CUD data

Authorisation to access CUD data should be requested by sending email to cud@oucs.ox.ac.uk, specifying the service principal that should be authorised to access the data.

5 Usage

Once you have the service principal you can generate a keytab using the kadmin utility on a workstation that has been secured for this purpose (since linux.ox.ac.uk is a shared service it is not recommended for this). Since the keytab contains password-equivalent keys for system-to-system authentication it should be kept in a secure location.

5.1 Using your Keytab

The following examples are based on a fictitious server called example.unit.ox.ac.uk, which is using a service principal,

```
cud/example.unit.ox.ac.uk@OX.AC.UK
```

for which a keytab has been created in the current directory with the name of

```
example.keytab
```

5.1.1 curl

The following is a simple example of using curl on a linux platform, authenticating with Kerberos to the CUD REST interface. Correct Kerberos configuration is assumed.

5.1.1.1 The Empty Query

```
$ KRB5CCNAME=krb5cc_test_cream kinit cud/example.unit.ox.ac.uk@OX.AC.UK -k -t example.keytab && \
        curl --negotiate -u : https://cud.oucs.ox.ac.uk/CoreUserDirectoryWs/rest/search?q=
```

5.1.2 NegotiateRestClient.jar

In contrast to curl, the java client uses a configuration file in which the principal and the keytab location are specified.

Given the example principal and keytab described above we will place our authentication configuration file in the current directory, and call it

```
login.conf
```

Its contents will look like the following:

```
gssnegotiate-keytab-client {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    keyTab="example.keytab"
    principal="cud/example.unit.ox.ac.uk";
};
```

NOTE that the values assigned to the attributes "keyTab" and "principal" above will need to be changed for your own set-up, i.e. you should substitute the correct values for the location and name of your own keytab file, and the name of your cud service principal.

With regard to the other configuration attributes, it is important that "useKeyTab" should be set to "true", to ensure that Java uses the keytab file to authenticate to the remote CUD REST interface.

To test the set-up it will help to have a basic kerberos configuration file in place so that the client has all its configuration information available locally. For our example this will be called

```
krb5.conf
```

By default the program will look in the current directory for the krb5.conf file. If the file is not found there Java will try to locate this file in these locations on a Windows platform (trying each in turn):

1. %JAVA_HOME%/lib/security/krb5.conf
2. %WINDOWS_ROOT%/krb5.ini

Alternatively you can specify the location of the Kerberos configuration file on the java command line, by setting the java.security.krb5.conf parameter, as in

```
% java -Djava.security.krb5.conf=/path/to/krb5.conf ...(etc)...
```

For use with the Oxford realm the contents of the krb5.conf kerberos configuration file should look like the following:

```
[libdefaults]
        default_realm = OX.AC.UK
[realms]
        OX.AC.UK  = {
                kdc = kdc0.ox.ac.uk
                kdc = kdc1.ox.ac.uk
                kdc = kdc2.ox.ac.uk
                kdc = kdc3.ox.ac.uk
                default_domain = OX.AC.UK
}
[domain_realm]
        .OX.AC.UK = OX.AC.UK
```

The krb5.conf above can be re-used as-is.

So to summarise, for this example we should have three files available: the keytab file (example.keytab), the authentication configuration file (login.conf), and the kerberos configuration file (krb5.conf). With that in mind the following are examples of using NegotiateRestClient.jar, and authenticating transparently with Kerberos to the CUD REST interface.

5.1.2.1 The Empty Query

The query is specified using the -d/--data flag, and the arguments following will be appended to the URL in a GET request. In this case, "q=" indicates that the query string is empty.

```
% java -Djava.security.krb5.conf=krb5.conf -jar NegotiateRestClient.jar -c login.conf -m gssnegotiate-keytab-
client -G -u https://ws.cud.ox.ac.uk/cudws/rest/search -d "q="
```

5.1.2.2 Some Example Queries

The following query retrieves all records that have a currently-active affiliation to OUCS, in XML format:

```
% java -Djava.security.krb5.conf=krb5.conf -jar NegotiateRestClient.jar -c login.conf -m gssnegotiate-keytab-
client -G -u https://ws.cud.ox.ac.uk/cudws/rest/search -d "q=cud\:cas\:current_affiliation:oucs&format=xml"
```

The next query retrieves a particular record for a person, with change history, in json format:

```
% java -Djava.security.krb5.conf=krb5.conf -jar NegotiateRestClient.jar -c login.conf -
m gssnegotiate-keytab-client -G -u https://ws.cud.ox.ac.uk/cudws/rest/search -d "q=cud\:cas
\:sso_username:unit1234&format=json&history=y"
```