Installing Shibboleth 1.3 on Redhat AS 3.0 using pubcookie for authentication

Table of Contents

Introduction

Shibboleth is a project conceived by a consortia of American educational institutes (Internet2/MACE) to solve the problem of authentication of users to resources provided by lots of external companies and institutes. For more details see http://shibboleth.internet2.edu/

Purpose

The purpose of this document is to lower the skill set required to install Shibboleth . This document follows the previous guide: 'Installing Pubcookie on Redhat AS 3.0 and authenticating against Windows Active Directory' available from http://iamsect.ncl.ac.uk/.

It is not the intention of this guide to give a full overview of how to install Redhat AS3.0. The Redhat installation documentation is good and should be used. This guide will focus on administration from the command line as most web servers run "headless" (i.e. no monitor, no windows, no graphical interface) so are only administrable by command line. Command line is available in all graphical environments.

About Shibboleth

Shibboleth is a single sign on project that has come out of a federation of higher education establishments in the United States called Internet2. It is now being viewed as the replacement for Athens login in the U.K.

Skill Sets required

In order to Install and manage a pubcookie server you will need to be able to access the following skills:

1. A reasonable working knowledge of the Linux (or Unix) Command Line Interface (CLI);
2. Knowledge of how to use the apache web server, either the 2.0.* or 1.3.* versions;
3. Familiarity with the concepts of https communication (certificates, keys and the like);
4. Familiarity with fire walls or access to someone who is familiar, in particular with Linux iptables style firewall;
5. Most importantly a willingness to read around subject areas, man pages, google, and mailing lists.

Equipment required

In order to be able to install a Shibboleth origin (identity provider) you will require the following:

1. A subscription to the Redhat network for at least one slot for Redhat AS3.0;
2. A standard "x86" server (i.e. A standard server with standard Intel or AMD chip set as sold by Dell etc);
3. The ability to synchronise system time against a network time protocol (NTP) server;
4. A SSL server certificate for the web server, the certificate must be valid and be signed by Certificate Authority (CA) e.g. Thawte or Verisign;
5. Root user access to the server;
6. A preexisting pubcookie login server installation as detailed in a previous document.

Note

While this install guide is based on an install on Redhat AS3.0, much of it is still applicable to other Linux distributions.

Assumptions

For the purpose of this guide we will assume that your institution has a web presence called "example.edu" and that your Shibboleth server will be called "shib.example.edu". We will also assume that you call your secure serving SSL key `shib.example.edu.key` and you call the signed certificate file you get back from your certificate authority `shib.example.edu.crt.signed`

Document Conventions

Code listings are provided within boxes like this:

```
$ echo hello world
```

File contents are provided within boxes like this:

```
The contents of a file
```

Some commands can be executed as a normal user, others are required to be executed with super-user privileges. You can follow this guide entirely as the super-user, or you could be a normal user for a number of steps. For the latter case, switching between users and managing resulting permission issues is an exercise left to the reader.

To indicate which category a given command falls into, the standard bourne-shell delimiters are used, '$' for normal user, '#' for superuser:

```
$ echo normal user command
# echo super user command
```

Installation

Installation of Shibboleth relies on the installation of several packages, applications and proper firewall configuration. While none of the steps are difficult in themselves, the combination of so many steps can be quite daunting. Hopefully the guide below will break down the installation process and ease the job of installing Shibboleth.

Prior Configuration

Remote File Systems (e.g. NFS)

Unless otherwise stated, do not build and install the components in this guide under a remote branch of the file system. All the examples here are installed under `/usr/local`. If `/usr/local` is part of a network-mounted or shared file system, replace this with something local, e.g. `/opt`.

Accessing the Web

Many of the following stages require obtaining files from the World Wide Web. If necessary, configure your environment so that tools such as wget are able to access the web:

```
$ export http_proxy=http://webproxy.example.edu:8080
```

Where webproxy.example.edu is the host and 8080 the port of your web proxy.

Firewall settings

It is generally regarded as good practice to have a firewall installed on servers. Redhat AS3.0 comes with the "iptables" firewall solution. If you have your own firewall solution or policy then you may wish to ignore this section.

It is probably a good idea to get the firewall setting sorted out as early as possible, as it is the kind of thing you can forget later on in an install. Since it drops connections silently it is difficult to identify as the source of a later problem.

In order to use Pubcookie you will need to allow:

1. network time protocol, this usually occurs via UDP communication to port 123;
2. HTTP and HTTPS web serving, the default ports are TCP 80 and TCP 443;
3. an additional port of 8443 is also required (more later), communication is via TCP;

4. for testing the Tomcat install opening up port 8080 to TCP is a good idea though in a live deployment you may wish to shut this down.

Tip

It is possible to check what is listening on an open port using `lsof -i:<port number>`

There are lots of ways to setup firewalls and lots of situations which require different firewall settings. If you don't know how to setup a firewall in your institution now is the time to consult someone who does. However, if you are just experimenting, the following rules would form the basis of a reasonable setup. Redhat keeps the iptables firewall rules in `/etc/sysconfig/iptables`. The syntax of this file is that generated by the tool `iptables-save`.

```
# iptables -A INPUT -p tcp -m tcp --dport 80 -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
# iptables -A INPUT -p udp -m udp --dport 123 -j ACCEPT
# iptables -A INPUT -p udp -m udp --dport 22 -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 8443  -j ACCEPT
# iptables -A INPUT -p tcp -m tcp --dport 8080 -j ACCEPT
```

The following commands ensure that local and existing connections are uninhibited:

```
# iptables -A INPUT -m state --state RELATED -j ACCEPT
# iptables -A INPUT -m state --state ESTABLISHED -j ACCEPT
# iptables -A INPUT -i lo -m state --state NEW -j ACCEPT
```

The following commands ensure that any traffic not already dealt with by the above commands will be dropped:

```
# iptables -P FORWARD DROP
# iptables -P INPUT   DROP
```

Caution

If you are remotely manipulating the machine, you may want to allow through the traffic that lets you do so. The secure shell protocol ssh operates on TCP port 22 and a rule is included above for it. Make sure you include this rule or one suitable for your setup before changing the default policy to 'DROP', especially if the machine is not physically close to you!

Note

There is some excellent documentation on iptables and its parent project netfilter at http://www.netfilter.org/. Go there for help writing more complex iptable recipes, including how to restrict access to individual addresses or subnets.

NTP setup

Both Shibboleth and Pubcookie requires accurate network time. They both use time stamps on cookies, forms, and SOAP messages to try and avoid "replay" attacks. In order to get accurate network time running it is a good idea to us xntpd to keep track of accurate time. Xntpd is a daemon that talks to a variety of time sources (as configured in ntp.conf) which can include specialised hardware clocks but in most case are just NTP servers which are in turn synchronised to accurate clocks. Running xntpd gives a much more stable and accurate time as it learns how inaccurate the local machine's clock is and can keep it reasonably well adjusted even if it loses contact with the remote servers.

Note

It may be possible to use a publicly accessible network time protocol server if you don't have access to

one. The web site at http://www.eecis.udel.edu/~mills/ntp/clock2a.html may provide information about this, though use at you own risk.

To configure edit `/etc/ntp.conf` to tell it which NTP server to use. You can put in multiple NTP server entries for redundancy.

```
#
# Standard NTP configuration for  systems
#

server  ntpA.example.edu
server  ntpB.example.edu
server  ntpC.example.edu

#
# Drift file.  Put this in a directory which the daemon can write to.
# No symbolic links allowed, either, since the daemon updates the file
# by creating a temporary in the same directory and then rename()'ing
# it to the file.
#
driftfile /etc/ntp/drift
```

Then to get xntpd to run and survive server reboots you need to:

```
# chkconfig ntpd on
# service ntpd start
```

To check it is running properly run:

```
$ ntpq -p
```

You should see output something like:

```
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
+ntp1.example.edu  ntp1.ja.net      2 u  605 1024  377    0.771   -0.796   0.170
 101.101.101.101   0.0.0.0         16 u    - 1024    0    0.000    0.000 4000.00
*ntp2.example.edu  ntp1.ja.net      2 u  298 1024  377    3.608   -2.041   1.324
 LOCAL(0)          LOCAL(0)        10 l   28   64  377    0.000    0.000   0.008
```

The above shows a working connection to the first and third ntp servers the second server is unreachable. A jitter of 4000.0 means the server is unreachable as does a reach of 0. You only need to be able to contact one server in order to sycronise, contacting more just makes it more resilient. It is also worth nothing that on some (possibly all) configs your clock needs to be reasonably close to accurate time in order to initially syncronise. Use the "date" utility to set the machine clock to something close to accurate time before setting up ntpd.

Apache Installation

In order to install Pubcookie you will require the httpd-devel version of Apache httpd as supplied by Redhat. The "devel" version is required as the apxs script is needed for the Pubcookie install. Obviously openssl and mod_ssl are required to provide secure serving necessary for logins.

To install the required packages simply run up2date:

```
# up2date httpd
```

```
# up2date httpd-devel
# up2date openssl
# up2date openssl-devel
# up2date mod_ssl
```

This will get you Apache 2, Apache development files including the module installer (apxs), openssl for secure SSL and mod_ssl to enable SSL in Apache.

It is a good idea to de-activate Apache modules that are not being used, to limit the amount of code that is being executed. The less code executed, the fewer potential security holes that are open to crackers.

If you are using the server for another purpose then you may need additional modules. De-activate other modules by editing `/etc/httpd/conf/httpd.conf`. We also suggest removing files (or renaming to remove the `.conf` extension) in `/etc/httpd/conf.d` which are unnecessary: e.g. `php.conf`, `perl.conf`, `python.conf`.

This step can probably be ignored if you are just testing the setup or think this is overly paranoid.

The following list of LoadModule directives will provide the required modules:

```
LoadModule access_module modules/mod_access.so
LoadModule auth_module modules/mod_auth.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule env_module modules/mod_env.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule mime_module modules/mod_mime.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule dir_module modules/mod_dir.so
LoadModule alias_module modules/mod_alias.so
LoadModule cgi_module modules/mod_cgi.so
```

Note

The modules `mod_ssi.so` `mod_jk2.so` and `mod_pubcookie.so` are also required, but they are inserted into `/etc/httpd/conf.d/ssl.conf` in the later stages of this guide.

Start the Apache httpd server as a service (for details of service script see /etc/init.d/httpd):

```
# service httpd start
```

Now browse to the server via http (visit http://weblogin.example.edu/) and https (visit https://weblogin.example.edu/). In both cases you should see the "Redhat Enterprise Linux test page".

Optionally you can use run levels to make httpd fire up after reboot using chkconfig (for full details use "man chkconfig" and Redhat documentation). In this case the server runs at run level 2,3 so httpd will start once the server reaches those run levels. Those running graphical (non headless) systems may wish to set it so httpd also runs at the run level required by these systems.

```
# chkconfig --level 23 httpd on
# chkconfig --list
```

You may wish to decide on a log rotation and retention policy. The default Redhat install will result in log file that are rotated weekly and kept for a month. This might not be ideal if you want to later looks at stats. In version 1.3 Shibboleth tries as much as possible to use it's own log files however the web server

logs may be useful especially for looking at errors due to probelms with ssl connections.

Getting proper signed certificates

It is necessary to generate proper signed certificate and key pair in order to work with Pubcookie. This needs to be signed by a trusted Certificate Authority as, in the secure negotiation when generating a key pair, Pubcookie checks the signing chain to make sure that the requesting server is valid.

```
$ openssl genrsa -out shib.example.edu.key 1024
$ openssl req -new -key shib.example.edu.key -out shib.example.edu.key.csr
```

You should then send the csr file to your Certificate Authority (Thawte, Verisign or similar) so that they can give you a fully signed certificate for use.

For testing you can generate a temporary self signed certificate until you get the fully signed one back from your Certificate Authority:

```
$ openssl x509 -req -days 30 -in shib.example.edu.csr -signkey shib.example.edu.ke
```

This certificate should be good enough for some light testing but you will have to wait until you get the signed one back in order to be able to use Pubcookie.

Note

There is a Makefile at /etc/httpd/conf which will allow you to generate certificates and key pairs however I prefer to do it by hand as at least then I know what I have done.

Pubcookie installation

As mentioned previously this guide assumes you have already setup a Pubcookie login server elsewhere. If not look at the companion guide 'Installing Pubcookie on Redhat AS 3.0 and authenticating against Windows Active Directory', accessible at http://iamsect.ncl.ac.uk/. It is not strictly necessary to use Pubcookie, however it is this guide's preferred choice of login provider. It is possible to use other techniques to allow people to login, mod_auth_ldap, Yale's CAS or even simple htpasswd files will all work. The important thing is that you have a mechanism for getting someones username into the REMOTE_USER server variable in the Apache httpd server. This is the recommended way of propagating authentication in Apache, so most of the modules should do this.

Installing the Pubcookie module

In order to get login you must install the Pubcookie authentication module. This is easily done using the steps below:

```
$ mkdir /usr/local/pubcookie
$ cd /usr/local/pubcookie
$ wget http://www.pubcookie.org/downloads/pubcookie-3.1.1.tar.gz
$ tar -xzvf pubcookie-3.1.1.tar.gz
$ cd pubcookie-3.1.1/
$ ./configure --prefix=/usr/local/pubcookie \
              --enable-apache               \
              --with-apxs=/usr/sbin/apxs
$ make
# make install
```

To confirm the module has installed:

```
ls -l /etc/httpd/modules
```

it should list mod_pubcookie.so as having been recently created. You will need to edit your /etc/httpd/conf.d/ssl.conf and /etc/httpd/conf/httpd.conf so that the module is loaded and configured properly.

Firstly edit /etc/httpd/conf.d/ssl.conf and where it says LoadModule ssl_module modules/mod_ssl.so append LoadModule pubcookie_module modules/mod_pubcookie.so so that it should read:

```
LoadModule ssl_module modules/mod_ssl.so
LoadModule pubcookie_module modules/mod_pubcookie.so
```

You will also need to configure a virtual host block to serve secure pages:

```
<VirtualHost _default_:443>
    DocumentRoot "/var/www/html"
    ServerAdmin webmaster@example.edu
    ServerName shib.example.edu:443
    # Use separate log files:
    ErrorLog logs/ssl_error_log
    TransferLog logs/ssl_access_log
    #   SSL Engine Switch:
    #   Enable/Disable SSL for this virtual host.
    SSLEngine on

    PubcookieAuthTypeNames EGNetID
    PubcookieGrantingCertFile /usr/local/pubcookie/keys/pubcookie_granting.cert
    PubcookieSessionKeyFile /etc/httpd/conf/ssl.key/shib.example.edu.key
    PubcookieSessionCertFile /etc/httpd/conf/ssl.crt/shib.example.edu.crt.signed
    PubcookieLogin https://shib.example.edu/cgi-bin/index.cgi
    PubcookieDomain .example.edu
    PubcookieKeyDir /usr/local/pubcookie/keys/
    ## Disable inactivity timeout by default to not timeout
    <Directory "/var/www/html">
        PubcookieInactiveExpire -1
    </Directory>

    <Directory "/var/www/html/test">
        AuthType EGNetID
        require valid-user
    </Directory>
</VirtualHost>
```

It is also necessary to edit /etc/httpd/conf/httpd.conf so that it contains the lines below. I add them just before the virtual hosts block. I'm not sure why it is necessary to duplicate these line in the httpd.conf, I would have thought them being in ssl.conf would be enough, however if PubcookieAuthTypeNames is not set here the server will fail to start, issuing a complaint that PubcookieAuthTypeNames is not set:

```
PubcookieAuthTypeNames EGNetID
PubcookieGrantingCertFile /usr/local/pubcookie/keys/pubcookie_granting.cert
PubcookieSessionKeyFile /etc/httpd/conf/ssl.key/shib.example.edu.key
PubcookieSessionCertFile /etc/httpd/conf/ssl.crt/shib.example.edu.signed
PubcookieLogin https://weblogin.example.edu/cgi-bin/index.cgi
PubcookieDomain .example.edu
PubcookieKeyDir /usr/local/pubcookie/keys/
```

If you now make a directory called test at /var/www/html/test

```
$ mkdir /var/www/html/test
```

and place a suitable index.html file into it (one saying "hello" will do). This can be used later for testing that the Pubcookie module is working.

Setting up the application server

To recap, the installation involves installation of Apache and the Pubcookie module, allowing communication over required ports (see firewall section above), setup of NTP and editing the Pubcookie config file. Make changes to the Apache config as described above. This should result in a working Pubcookie application server. The only difference is that you need to transfer the pubcookie_granting.cert file to the application server (default location is /usr/local/pubcookie/keys/pubcookie_granting.cert) vi sftp or whatever mechanism you are comfortable with. Then tell the keyserver on the login server to accept connections from this application server . On the login server:

```
$ cd/usr/local/pubcookie
$ ./keyclient -P shib.example.edu
```

You may also need to add shib.example.edu to the appropriate section of the hosts.allow file as detailed in the previous document. To briefly recap here it would be something like:

```
keyserver: shib.example.edu
```

Then request a key from the application server:

```
$ cd/usr/local/pubcookie
$ ./keyclient
make_crypt_keyfile: hello
make_crypt_keyfile: goodbye
Set crypt key for shib.example.edu
```

If this fails look through the logs in `/var/log/secure` and `/var/log/messages` on both machines to get error messages. If successful this should result in a fully working application server.

Note

It should be possible to set up the appserver and login server to distribute the pubcookie_granting cert automatically however attempts to do this failed. Similarly it should not be necessary to use the permit (-P) option on the login server as the keyserver_client_list should inform it which server can request keys. However in practice this step proved to be necessary

Warning

As described above in 'Remote File Systems (e.g. NFS)', Pubcookie directories should not be mounted on NFS. This is due to the security model built into the Pubcookie design: if a cracker takes control of an application server, they haven't automatically compromised the other application servers, or the authentication server. However if they all shared all the keys and certificates compromising one application server would lead to communication with all being untrustworthy.

Restart the server to make sure that changes are recognised:

```
# service httpd restart
```

then browse to the test directory using a web browser e.g. go to https://shib.example.edu/test/index.html

You should be redirected to the login server you setup previously where you can login, on successful login you should be redirected back to the index.html page e.g. https://shib.example.edu/test/index.html. If there are problems check the Apache error and access logs and check in /var/log/secure and /var/log/messages for potential causes. It may be necessary to do this both on the application server and the login server. If you were successful you now have a means of getting a user's login name into the REMOTE_USER server variable and can start thinking about getting Shibboleth working.

Installing and configuring Java

Download from http://java.sun.com/products/archive/ the Java2 SDK version 1.4.2_04 (j2sdk-1_4_2_04):

Note

there where bugs reported in shibboleth 1.2 and j2sdk-1_4_2_05, i'm unsure if this has been rectified in shibboleth 1.3. It should be possible to use more modern versions of the Java JDK with shibboleth 1.3 as no serious problems have been reported on the mailing lists however the author can't confirm this as we haven't tested.

```
$ sh ./j2sdk-1_4_2_04-linux-i586-rpm.bin
$ rm ./j2sdk-1_4_2_04-linux-i586-rpm.bin
# rpm -i j2sdk-1_4_2_04-linux-i586.rpm
```

This should have installed Java at /usr/java/j2sdk1.4.2_04. You now need to setup the JAVA_HOME environment variable so that it points to Java and Java based applications know where to find the Java commands. You will probably have your own favourite way of doing this, for those new to this the two scripts below would make sure bash and c shell have JAVA_HOME set to the jdk you installed. If you later decide to upgrade the jdk you should change the script to reflect the new location.

Edit /etc/profile.d/java.csh so it contains:

```
if ( "${path}" !~ */usr/java/j2sdk1.4.2_04/bin* ) then
        set path = ( /usr/java/j2sdk1.4.2_04 $path )
endif
setenv JAVA_HOME /usr/java/j2sdk1.4.2_04
```

Edit /etc/profile.d/java.sh so it contains:

```
if ! echo ${PATH} | grep -q /usr/java/j2sdk1.4.2_04/bin ; then
        PATH=/usr/java/j2sdk1.4.2_04/bin:${PATH}
fi
JAVA_HOME="/usr/java/j2sdk1.4.2_04"
export JAVA_HOME
```

To test whether this has worked logoff and log back in (so that your shell picks up the changes) and then use the 'which' command:

```
$ which java
/usr/java/j2dsk1.4.2_04/bin/java
$ which javac
/usr/java/j2dsk1.4.2_04/bin/javac
```

If this hasn't worked, either the scripts that set Java into your path and environment variables have failed or the Java install has failed. You should check your installation.

Installing Apache Tomcat

You need to grab a version of Apache Tomcat. This is available from Apache software mirrors, a list of which is available at http://www.apache.org/dyn/closer.cgi.

We used http://apache.mirror.positive-internet.com/, but the URL above will suggest one to you. Substitute your particular mirror in the commands below.

Tomcat is kept within the directory jakarta/tomcat-5/. At the time of writing version 5.0.27 was available and so we used that. However Apache do not keep older versions of software available for long so you may not be able to find this version. 5.0.28 is reported to work too. Substitute the version number below for one that is available:

```
$ mkdir /usr/local/tomcat
$ cd /usr/local/tomcat
$ wget http://apache.mirror.positive-internet.com/jakarta/tomcat-5/v5.0.27/bin/jak
$ tar -zxf jakarta-tomcat-5.0.27.tar.gz
$ tar -zxf Fedora-Core-1-i386.tar.gz
```

In order to make upgrading easy I create a symlink from /usr/local/tomcat/current to the version of Tomcat I am running. This makes upgrading easier and more importantly makes backing out of a bad upgrade easier.

```
$ ln -s jakarta-tomcat-5.0.27 current
$ cd current/bin
```

For the sake of tidiness and so files are easy to see I remove some of the Windows specific control files (if you envisage moving to Windows then you may wish to leave them)

```
$ rm *.bat
```

You should now test that Tomcat can run:

```
$ cd /usr/local/tomcat/current/bin
$ ./startup.sh
```

To verify it is running, you can browse to http://shib.example.edu:8080/. You should get a Tomcat page that tells you it has installed and lists a couple of test servlets and JSPs.

Alternatively you could execute the command:

```
$ ps -ef | grep tomcat
```

Which should list a process with a command containing '/usr/java/j2sdk1.4.2_04/bin/java -Djava.endorsed.dirs=/opt/tomcat/current/common/endorsed ...'

If this doesn't occur look through the Tomcat logs at /usr/local/tomcat/current/logs to find possible reasons.

If the test is successful shutdown the server using the command:

```
$ ./shutdown.sh
```

Note

you can install on tomcat 5.5 but you will probably use java 1.5 (branded as 5.0 i think) as tomcat 5.5 is designed for that platform

Install the mod_jk2 connector

Now that you have a working Tomcat and Apache install you need to install the mod_jk2 connector so that Apache can pass the authenticated user name from Pubcookie over to Tomcat so that is can be used by Shibboleth later on. To do this you must download the appropriate mod_jk2 connector binary and install it. Good sources are rpmfind.net or http://www.jpackage.org/, jpackage will probabably have redhat AS specific rpms, select which one you are confortable with and ddownload and install the rpm e.g. :

```
$ wget \
ftp://fr2.rpmfind.net/linux/fedora/core/2/i386/os/Fedora/RPMS/mod_jk2-4.1.27-13.i3
# rpm -i mod_jk2-4.1.27-13.i386.rpm
```

Note

If you do not trust rpmfind.net or jpackage to provide safe binaries, you can obtain the source code to mod_jk2 from http://jakarta.apache.org/site/sourceindex.cgi#tomcat-connectors , verify the source and build it yourself.

To confirm it has installed you should use rpm's ability to query a package, which should print out the location of files from the RPM on the filesystem:

```
$ rpm -ql mod_jk2
/etc/httpd/conf.d/jk2.conf
/etc/httpd/conf/workers2.properties
/usr/lib/httpd/modules/mod_jk2.so
/usr/share/doc/mod_jk2-4.1.27
/usr/share/doc/mod_jk2-4.1.27/CHANGES.html
/usr/share/doc/mod_jk2-4.1.27/CHANGES.txt
/usr/share/doc/mod_jk2-4.1.27/README.txt
/usr/share/doc/mod_jk2-4.1.27/STATUS.txt
```

Because the jk2.conf file is in the conf.d directory it will now be included by Apache upon restart, this will load the mod_jk2.so module into Apache. Since jk2.conf also references the workers2.properties file it to will also be recognised by Apache after a restart.

It is now a good idea to test the mod_jk connector is working.

Replace the file /etc/httpd/conf/workers2.properties by one containing the following text:

```
[shm]
info=Scoreboard. Required for reconfiguration and status with multiprocess servers
file=anon
# Defines a load balancer named lb. Use even if you only have one machine.
[lb:lb]
# Example socket channel, override port and host.
[channel.socket:localhost:8009]
port=8009
host=127.0.0.1
# define the worker
[ajp13:localhost:8009]
channel=channel.socket:localhost:8009
group=lb
# Map the Tomcat examples webapp to the Web server uri space
[uri:/jsp-examples/*]
group=lb
[uri:/shibboleth/*]
worker=ajp13:localhost:8009
group=lb
```

```
[status:]
info=Status worker, displays runtime information

[uri:/jkstatus/*]
info=The Tomcat /jkstatus handler
group=status:
```

To summarise this rather dense configuration: this file tells the mod_jk2 connector to create a socket that listens on port 8009 on local host, it then assigns that socket to a mod_jk worker and adds that worker to a group called lb (in this case it is a group of 1 but in other configs you can have multiple workers in a group). The group lb will then be used take any requests to /jsp-examples or /shibboleth and redirect them to Tomcat.

Make sure that both Tomcat and httpd have been restarted, you should then try browsing to http://shib.example.edu/jsp-examples/, this should get you the same page as browsing directly to Tomcat http://shib.example.edu:8080/jsp-examples/. If this works then you have successfully used the mod_jk2 connector to mount Tomcat served webapps in Apache httpd. The entry in the workers2.properties file for Shibboleth will do nothing at this point, however it will be used later on in this install guide.

Stop Tomcat and edit the /usr/local/tomcat/current/conf/jk2.properties file so that it contains the line:

```
request.tomcatAuthentication=false
```

This will force Tomcat to take its authentication from httpd and will allow you to pass the username Pubcookie stores in REMOTE_USER into Shibboleth when it is configured.

Installing Shibboleth

Now that you have a working httpd server which can authenticate users and you can pass that authentication to Tomcat you are now in a position to deploy Shibboleth.

First it is necessary to download the Shibboleth origin (identity provider) install:

```
$ cd /opt
$ wget http://shibboleth.internet2.edu/downloads/shibboleth-idp-1.3c.tar.gz
$ tar -zxvf shibboleth-idp-1.3c.tar.gz
```

next you need to build and compile shibboleth using the java tool ant. Ant is a java program a bit like make, it's not really important to know what it is or how it works just that in installs shibboleth. Ant relies on having your JAVA_HOME environment variable set properly so it may complain if you havne't done that properly (this gude has already told you how)

```
cd shibboleth-idp-1.3c-install/
./ant
```

The build script will ask a series of questions about what and where to install The screen input blow will get you a working install

```
Do you want to install the Shibboleth Identity Provider? [Y,n] Y
What name do you want to use for the Identity Provider web application? [default:

Deploying the java web application. Do you want to install it directly onto the fi
1) filesystem (default)
2) manager
```

```
1

Select a home directory for the Shibboleth Identity Provider [default: /usr/local/
Enter the tomcat home directory [default: /usr/local/tomcat] /usr/local/tomcat/cur
```

It is also necessary to copy several jar files into an endorsed directory. The reason for this is that the Java you installed earlier comes with XML tools that are too old to work with Shibboleth so you have to override them by placing newer ones in Tomcat's endorsed directory:

```
$ rm /usr/local/tomcat/current/common/endorsed/xercesImpl.jar
$ rm /usr/local/tomcat/current/common/endorsed/xml-apis.jar
$ cp /opt/shibboleth-idp-1.3c-install/endorsed/*.jar /usr/local/tomcat/current/com
```

Restart Tomcat and it should automatically expand the shibboleth.war file deployed by the install script above to /usr/local/tomcat/current/webapps/shibboleth/:

```
$ cd /usr/local/tomcat/current/bin
$ ./shutdown.sh
$ ./startup.sh
```

In order to overcome a bug in Apache httpd 2.0 it is necessary to mount the Attribute Authority on a separate virtual host from the rest of the Shibboleth install. Edit your /etc/httpd/conf.d/ssl.conf file so that it has entries something like the one below. The important thing is to make sure that the <Location "/shibboleth/HS"> is there and protected by Pubcookie so that people have to login to access the handle service. It is also important that the <Location "/shibboleth/AA"> is in a separate virtual host to overcome the Apache bug. The easiest way of doing this is to stick a virtual host on the port 8443, that way you can use the same certificates (SSL certs just verify the server name e.g. shib.example.edu, they don't "care" about which port is being used) to do this you can edit the /etc/httpd/conf.d/ssl.conf file so it looks something like this:

```
<VirtualHost _default_:443>
    DocumentRoot "/var/www/html"
    ServerAdmin webmaster@example.edu
    ServerName shib.example.edu:443
    ErrorLog logs/ssl_error_log
    TransferLog logs/ssl_access_log
    SSLEngine on
    PubcookieAuthTypeNames EGNetID
    PubcookieGrantingCertFile /usr/local/pubcookie/keys/pubcookie_granting.cert
    PubcookieSessionKeyFile /etc/httpd/conf/ssl.key/shib.example.edu.key
    PubcookieSessionCertFile /etc/httpd/conf/ssl.crt/shib.example.edu.crt.signed
    PubcookieLogin https://shib.example.edu/cgi-bin/index.cgi
    PubcookieDomain .example.edu
    PubcookieKeyDir /usr/local/pubcookie/keys

    <Directory "/var/www/html">
        PubcookieInactiveExpire -1
    </Directory>

    <Directory "/var/www/html/test/">
        AuthType EGNetID
        require valid-user
    </Directory>

    <Location "/shibboleth/HS">
        AuthType EGNetID
        require valid-user
```

```
    </Location>

    SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP
    SSLCertificateFile /etc/httpd/conf/ssl.crt/shib.example.edu.crt.signed
    SSLCertificateKeyFile /etc/httpd/conf/ssl.key/shib.example.edu.key
    SSLCACertificateFile /usr/share/ssl/certs/ca-bundle.crt
    SetEnvIf User-Agent ".*MSIE.*" \
             nokeepalive ssl-unclean-shutdown \
             downgrade-1.0 force-response-1.0
    CustomLog logs/ssl_request_log \
              "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
</VirtualHost>

<VirtualHost _default_:8443>
    LogLevel info
    ErrorLog /usr/local/shibboleth-idp/logs/AAssl_error_log
    TransferLog /usr/local/shibboleth-idp/logs/AAssl_access_log
    SSLEngine on
    SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP
    #   Server Certificate:
    SSLCertificateFile /etc/httpd/conf/ssl.crt/shib.example.edu.signed
    SSLCertificateKeyFile /etc/httpd/conf/ssl.key/shib.example.edu.key
    SSLCACertificateFile /usr/share/ssl/certs/ca-bundle.crt

    SSLVerifyClient optional
    SSLOptions +OptRenegotiate
    SSLOptions +StdEnvVars +ExportCertData

    <Location /shibboleth/AA>
        SSLVerifyClient optional
        SSLOptions +OptRenegotiate
        SSLOptions +StdEnvVars +ExportCertData
    </Location>

    SetEnvIf User-Agent ".*MSIE.*" \
             nokeepalive ssl-unclean-shutdown \
             downgrade-1.0 force-response-1.0
    CustomLog logs/AAssl_request_log \
             "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
</VirtualHost>
```

After the line:

```
Listen 0.0.0.0:443
```

Insert the line:

```
Listen 0.0.0.0:8443
```

To summarise, the above file configures httpd so that Pubcookie is configured and then protects the Shibboleth handle service "/shibboleth/HS" by making sure users have to be logged in via Pubcookie to access it. It then creates a separate virtual host on port 8443 and places the Shibboleth attribute authority on it. The virtual hosting of this on a separate virtual host is solely there to work around a known Apache httpd SSL bug.

changes in shibboleth 1.3

Shibboleth 1.3 has made some changes to the default location of configuration files. In 1.2 most of the files resided in a directory structure in the WEB-INF directory of the shibboleth webapplication. In 1.3 config files logs and some tools now reside in the default location of /usr/local/shibboleth-idp the main

config file has changed names from origin.xml to idp.xml and is located at /usr/local/shibboleth-idp/etc

Setup the idp.xml file

It is necessary to edit the /usr/local/tomcat/current/webapps/shibboleth/WEB-INF/classes/conf/origin.xml file to configure Shibboleth to suitable values for your origin. Below is an example configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<ShibbolethOriginConfig
        xmlns="urn:mace:shibboleth:origin:1.0"
        xmlns:cred="urn:mace:shibboleth:credentials:1.0"
        xmlns:name="urn:mace:shibboleth:namemapper:1.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:mace:shibboleth:origin:1.0 origin.xsd"
        AAUrl="https://shib.example.edu:8443/shibboleth/AA"
        resolverConfig="file:/usr/local/shibboleth-idp/etc/resolver.xml"
        defaultRelyingParty="urn:mace:inqueue"
        providerId="urn:mace:inqueue:example.edu">

<!-- You'll need to get a test credential from an InQueue CA -->

<RelyingParty name="urn:mace:inqueue" signingCredential="inqueue_cred">
  <HSNameFormat nameMapping="shm"/>
</RelyingParty>

<ReleasePolicyEngine>
 <ArpRepository implementation="edu.internet2.middleware.shibboleth.aa.arp.provide
   <Path>/conf/arps/</Path>
 </ArpRepository>
</ReleasePolicyEngine>

<Logging>
  <ErrorLog level="DEBUG" location="file:///tmp/shib-error.log" />
  <TransactionLog location="file:///tmp/shib-access.log" />
</Logging>

<NameMapping xmlns="urn:mace:shibboleth:namemapper:1.0" id="shm"
 format="urn:mace:shibboleth:1.0:nameIdentifier"  type="SharedMemoryShibHandle"
 handleTTL="1800"/>

<Credentials xmlns="urn:mace:shibboleth:credentials:1.0">

  <FileResolver Id="foo">
    <Key format="PEM">
      <Path>file:///etc/httpd/conf/ssl.key/shib.example.edu.key</Path>
    </Key>
    <Certificate format="PEM">
      <Path>file:///etc/httpd/conf/ssl.crt/shib.example.edu.crt.signed</Path>
    </Certificate>
  </FileResolver>

  <FileResolver Id="inqueue_cred">
    <Key format="PEM">
      <Path>file:///etc/httpd/conf/ssl.key/shib.example.edu.key</Path>
    </Key>
    <Certificate format="PEM">
      <Path>file:///etc/httpd/conf/ssl.crt/shib.example.edu.crt.signed</Path>
    </Certificate>
  </FileResolver>

</Credentials>
```

```
<FederationProvider
 type="edu.internet2.middleware.shibboleth.metadata.provider.XMLMetadataLoadWrappe
 uri="/conf/sites.xml"/>

</ShibbolethOriginConfig>
```

It is important to get AAUrl to point at the AA on your server remembering that it is on port 8443 e.g. AAUrl="https://shib.example.edu:8443/shibboleth/AA. Also make the providerId reflect what is required for your setup e.g. providerId="urn:mace:inqueue:example.edu". It is important that the File resolver element is set for the inqueue_cred as since it is referenced by the signingCredentials attribute of the RelyParty element shibboleth won't start without it. Also set the defaultRelyingParty to the InQueue federation. The Logging element has been configured to debug and will log to the /tmp directory so check there when things don't work.

Setting up the Attribute Authority (AA) configuration

It is also necessary to edit the resolver.xml file /usr/local/shibboleth-idp/etc/resolver.xml so that your AA knows how to pickup attributes from their stores, in this case we are going to add the simplest form where the AA will just echo no attributes. This is enough to get working with the test page for InQueue. At a later time it will be necessary to edit this so that actual attributes are fetched by the AA. The smart scope of the attributes should also be set as the domain name of the institute e.g. example.edu

```
<AttributeResolver
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="urn:mace:shibboleth:resolver:1.0"
 xsi:schemaLocation="urn:mace:shibboleth:resolver:1.0 shibboleth-resolver-1.0.xsd"

  <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonEntitlement">
    <DataConnectorDependency requires="echo"/>
  </SimpleAttributeDefinition>

  <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonAffiliation">
    <DataConnectorDependency requires="echo"/>
  </SimpleAttributeDefinition>

<!-- To use these attributes, you should change the smartScope value to match your
  <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonScopedAffilia
    <AttributeDependency requires="urn:mace:dir:attribute-def:eduPersonAffiliation
  </SimpleAttributeDefinition>

  <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonPrincipalName
    <DataConnectorDependency requires="echo"/>
  </SimpleAttributeDefinition>

  <CustomDataConnector id="echo" class="edu.internet2.middleware.shibboleth.aa.att
</AttributeResolver>
```

Normally you would now edit the attribute release policy files in the arps.site.xml in the directory /usr/local/tomcat/current/webapps/shibboleth/WEB-INF/classes/conf/arps/ in a real live install. However the attribute release policy that comes with Shibboleth are good enough for a test install. In our current case there are no attributes to release. In a real deployment you would need to edit this file to get a sensible policy for you institution.

Setting your atribute release policy (ARP)

It is also necessary to setup you attribute release policy (ARP) to release the attribute you have just con-

figured to service providers. It is impossible to write a comprehensive guide to this as it is very much dependant on an institutes policies and what attribute are going to be useful. For testing it is possible to write a configuration that releases your attributes to any service provider. However you should decide on your own policy and read the documentataion on the shibboleth site as to how to implement it. A specimen overly permissive release policy is listed below, it's alright for testing but should not be used on a live system.

```
<?xml version="1.0" encoding="UTF-8"?>
<AttributeReleasePolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" x
mlns="urn:mace:shibboleth:arp:1.0" xsi:schemaLocation="urn:mace:shibboleth:arp:
1.0 shibboleth-arp-1.0.xsd" >
        <Description>Simplest possible ARP.</Description>
        <Rule>
                <Target>
                        <AnyTarget/>
                </Target>
                <Attribute name="urn:mace:dir:attribute-def:eduPersonAffiliation">
                        <AnyValue release="permit"/>
                </Attribute>
                <Attribute name="urn:mace:dir:attribute-def:eduPersonScopedAffilia
                        <AnyValue release="permit"/>
                </Attribute>
                <Attribute name="urn:mace:dir:attribute-def:eduPersonTargetedID">
                        <AnyValue release="permit"/>
                </Attribute>
                <Attribute name="urn:mace:dir:attribute-def:eduPersonPrincipalName
                        <AnyValue release="permit"/>
                </Attribute>
                <Attribute name="urn:mace:dir:attribute-def:eduPersonScopedEntitle
                        <AnyValue release="permit"/>
                </Attribute>
                <Attribute name="urn:mace:dir:attribute-def:eduPersonEntitlement">
                        <AnyValue release="permit"/>
                </Attribute>
</Rule>
</AttributeReleasePolicy>
```

A guide to the format of arps can be found at http://shibboleth.internet2.edu/guides/idp/infoarps.html and a tool called SHARPE is being developed to allow user control of attribute release see http://mams.melcoe.mq.edu.au/wiki/display/MAMS/Shibboleth+Attribute+Request+Policy+Editor+(ShARPE) for details.

Join the InQueue federation

Now you have a technically working Shibboleth install you need to join a federation in order to test it. The American InQueue federation http://inqueue.internet2.edu/ is a good candidate at present. Fill in the form at http://inqueue.internet2.edu/join.html. You should sign up as an Identity Provider.

When filling in the form you will need to know:

1. The domain name of the Identity Provider site, this will be something like example.edu;
2. The complete URL for the handle service is e.g. https://shib.example.edu/shibboleth/HS;
3. The complete URL for the attribute authority , remember this is going to be virtual hosted onto port 8443 e.g.https://shib.example.edu:8443/shibboleth/AA;
4. The CN or the full subject of the HS's certificate, this should be the name of the server e.g. shib.example.edu;

5. The short name(s) the WAYF should display for this identity provider is basically this is what users are going to select in a drop down list when the tell the where are you from (WAYF) service which institute they are from so maker it recognisable and in keeping with your brand e..g "The University of Example".

Once you have joined a federation you will have to download the sites.xml. file that they provide, this file tells your Shibboleth install who else is in the federation so that Shibboleth can decide whether to trust them. It may be necessary to wait a day until the federation has updated your details into the file. Fortunately Shibboleth has provided a tool that can be automated by a cron job so that you don't have to manually download every time a new member joins the federation. In order to run the tool use the command below:

```
cd /usr/local/shibboleth-idp/
./bin/metadatatool -i http://wayf.internet2.edu/InQueue/IQ-sites.xml \
-k conf/internet2.jks -p shib123 -a sitesigner \
-o /usr/local/tomcat/current/webapps/shibboleth/WEB-INF/classes/conf/sites.xml
```

This should save the latest federation sites into /usr/local/tomcat/jakarta-tomcat-5.0.27/webapps/shibboleth/WEB-INF/classes/conf/sites.xml. It may be informative to have a look through the file.

Now that you have successfully joined go to https://wayf.internet2.edu/inQueue/sample.jsp since you aren't yet authenticated you should be redirected to a page showing a drop down menu of institutes select you own and you should be redirected to your home pubcookie login, on successful login you should be redirected back to the original https://wayf.internet2.edu/InQueue/sample.jsp page. If you manage to get to the page without it reporting errors then you have a working test install

Log files

Shibboleth is a realtively complex system with interoperating component parts. Often it is necessary to look in log files to work out configuration errors. Logging of the shibboleth processes is configure using the idp.xml file in /usr/local/shibboleth-idp/etc the lines below being the configuration.

```
<Logging>
    <ErrorLog level="DEBUG" location="file:/usr/local/shibboleth-idp/logs/shiberro
    <TransactionLog level="DEBUG" location="file:/usr/local/shibboleth-idp/logs/sh
</Logging>
```

There is more to a working shibboleth configuration than just the shibboleth process however, sometimes things appear in the apache logs that may be relevant. The section in the apache ssl config file at /etc/httpd/conf.d/ssl.conf

```
<VirtualHost _default_:8443>
    LogLevel info
    ErrorLog /usr/local/shibboleth-idp/logs/AA_ssl_error_log
    TransferLog /usr/local/shibboleth-idp/logs/AA_ssl_access_log
```

This will tell the virtual server that handle requests to the attribute authority section of shibboleth to log any errors to the smae directory as your shibboleth software. In this way most of the relevant log entires will be contained in one directory. The AA_ssl_error_log can be particularly infomative if you are having problems and nothing is appearing in the shibboleth logs. Wrong configuration of your ssl can lead to apache ssl dropping the connections before shibboleth ever gets a chance to deal with it. In a production deployment you may wish to alter logging locations and behavior to suit what your institution does, however in an initial install having the log files in one place can help.

Using Real Attributes

The configuration described aboove using the detault simple file whose use is configured by the line resolverConfig="file:/usr/local/shibboleth-idp/etc/resolver.xml" in idp.xml is reasonable for testing and in many cases may be suitable for your production deployment, if you just don't need to exchange complicated attributes. However many institute will need to use complicated attributes which riside in ldap directories or databases. Shibboleth Attribute Authority does support pulling attributes out of these stores in a basic way. It also supports custom connectors which can be written in java to enable institutes to obtain attributes in bespoke tailored ways using complex logic. Shibboleth comes bundeled with two connector configurations /usr/local/shibboleth-idp/etc/resolver.jdbc.xml and / usr/local/shibboleth-idp/etc/resolver.ldap.xml. The database connector proved to be the best choce for our institute.

Using the database connector

in order to use the jdbc connector to connect to databases you need to alter the idp.xml so that the resolver configt line now reads as resolverConfig="file:/usr/local/shibboleth-idp/etc/resolver.jdbc.xml" you can then edit the specimen /usr/local/shibboleth-idp/etc/resolver.jdbc.xml to connect to your databse and get the attribute you need. Below is an example file with example configs illustrating the main features available:

```
<AttributeResolver xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="ur
er:1.0 shibboleth-resolver-1.0.xsd">

<!--
example of a simple attribute eduPersonPrincipalName which is pulled from a data b
notw by default the returned value pair has to have the right name so i have to re
the username AS eduPerson principla name so the resolver knows to use it as this a
The ? is replaced by what was passed to shibboleth from the REMOTE_USER apache var
populated by whatever login programme you are using, in this case pubcookie.
-->

        <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonPrincip
                <DataConnectorDependency requires="db1"/>
        </SimpleAttributeDefinition>

        <JDBCDataConnector id="db1"
                dbURL="jdbc:mysql://database.example.edu/attributedatabase?user=so
                dbDriver="com.mysql.jdbc.Driver"
                maxActive="10"
                maxIdle="5">
                <Query>SELECT username AS eduPersonPrincipalName FROM usertable WH
        </JDBCDataConnector>

<!--end example of a simple attribute eduPersonPrincipalName  -->

<!--
example of a complex  attribute eduPersonTargetedID which is pulled from a data ba
as a randomised but consistent user identifier that service providers can use to p
without ever actually knowing who the user is. There is some controversy around ed
-->

 <PersistentIDAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonTargeted

             <DataConnectorDependency requires="db2"/>
             <Salt>put some random stuff here</Salt>
         </PersistentIDAttributeDefinition>
        <JDBCDataConnector id="db2"
                dbURL="jdbc:mysql://database.example.edu/attributedatabase?user=so
                dbDriver="com.mysql.jdbc.Driver"
```

```
                maxActive="10"
                maxIdle="5">
                <Query>SELECT username AS eduPersonTargetedID FROM usertable WHERE
        </JDBCDataConnector>

<!--end example of a complex  attribute eduPersonTargetedID  -->

<!--
example of a complex attribute eduPersonEntitlement which is pulled from a data ba
This shows it is poosible to encapsulate quite complex logic in the SQL query. How
this can get ugly and it may be that you need to contamplate coding a custom conne
the logic offerred by java rather than relying on your sql query to do it
Note sourceName="sdssentitlement" in the Definition blow means the resolver will l
column name so you don't have to name the column as the attribute name

This is similar to a query we use to identify if a student is a on a medical cours
from EMOL (autopsy videos and the like)
-->
     <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonEntitlemen
                <DataConnectorDependency requires="db3"/>
     </SimpleAttributeDefinition>

    <JDBCDataConnector id="db3"
                dbURL="jdbc:mysql://database.example.edu/attributedatabase?user=so
                dbDriver="com.mysql.jdbc.Driver"
                maxActive="10"
                maxIdle="5">
                <Query>SELECT course_code,
                       CASE course_code
                       WHEN 'A101' THEN 'urn:mace:ac.uk:sdss.ac.uk:entitlement:emo
                       WHEN 'A106' THEN 'urn:mace:ac.uk:sdss.ac.uk:entitlement:emo
                       WHEN 'O106' THEN 'urn:mace:ac.uk:sdss.ac.uk:entitlement:emo
                       WHEN '3019P' THEN 'urn:mace:ac.uk:sdss.ac.uk:entitlement:em
                       WHEN '3384P' THEN 'urn:mace:ac.uk:sdss.ac.uk:entitlement:em
                       WHEN '5826P' THEN 'urn:mace:ac.uk:sdss.ac.uk:entitlement:em
                       ELSE 'none' END
                       AS sdssentitlement FROM coursetable WHERE userid = ?
                </Query>
        </JDBCDataConnector>

<!--end example of a complex  attribute eduPersonEntitlement  -->


<!--
example of a dual query pulling back several attribute from a data base
note these are fake attribute for a fake example which i don't use
I don't have a need for this kind of query so i have not tested it.
the urn value are made up to and you should look up guides on how to make a proper
-->

        <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:role" >
                <DataConnectorDependency requires="db4"/>
        </SimpleAttributeDefinition>
        <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:group" >
                <DataConnectorDependency requires="db4"/>
        </SimpleAttributeDefinition>

        <JDBCDataConnector id="db4"
                dbURL="jdbc:mysql://database.example.edu/attributedatabase?user=so
                dbDriver="com.mysql.jdbc.Driver"
                maxActive="10"
                maxIdle="5">
                <Query>SELECT role, groupFROM usertable WHERE userid=?</Query>
```

```
        </JDBCDataConnector>

<!--end example of a simple attributes role and group  -->

<!--
example of an attribute that depends on the value of another. Also demonstarates t
because they are availabl in shibboleth without querying .
-->

        <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonScopedA
                <AttributeDependency requires="urn:mace:dir:attribute-def:eduPerso
        </SimpleAttributeDefinition>
        <SimpleAttributeDefinition id="urn:mace:dir:attribute-def:eduPersonAffilia
                <DataConnectorDependency requires="echo"/>
        </SimpleAttributeDefinition>

        <CustomDataConnector id="echo" class="edu.internet2.middleware.shibboleth.

</AttributeResolver>
```

Polishing the install

Updating Federation metadata regularly

Make sensible pages

Pubcookie login pages (mainly the login template) in the web app

IdP.jsp IdPError.jsp accessError.html

error pages and explanation on iss site

User education

Things to think about