

KC-ROLO Shibboleth Guide

V 1.0
23/07/2007



Shibboleth®

Authors:

**Ed Beddows
Tim Hall**

**Kidderminster College
KC-ROLO Project Funded by JISC**

JISC

<http://kidderminster.ac.uk/kc-rola>

Contents

INTRODUCTION.....	3
PURPOSE	3
PREREQUISITES	3
SHIBBOLETH OVERVIEW	4
WHAT IS IT?	4
HOW DOES IT WORK?	4
Origin (Identity Provider)	4
Target (Service Provider)	5
WAYF (Where Are You From?)	5
Message Flow	6
FEDERATIONS	8
WHAT IS A FEDERATION?	8
IDENTITY PROVIDER IMPLEMENTATION (V1.2.1).....	9
OPERATING SYSTEM	9
UPDATE PROCESS	9
NTP CONFIGURATION.....	9
FIREWALL CONFIGURATION	10
INSTALL APACHE AND ASSOCIATED FILES	11
INSTALL JAVA.....	11
INSTALL TOMCAT 5	11
INSTALL AND SETUP JK2 CONNECTOR.....	12
INSTALL SHIBBOLETH.....	13
CREATE CERTIFICATES	14
CONFIGURE APACHE CONFIGURATION FILES	14
EDIT ORIGIN.XML.....	15
EDIT SITES.XML	16
SETTING UP THE ATTRIBUTE RESOLVER	16
SETTING UP THE ATTRIBUTE RELEASE POLICY.....	18
USEFUL LOG FILES	19
SERVICE PROVIDER IMPLEMENTATION (V1.2.1A)	20
INSTALL WEB SERVER	20
INSTALLING OPENSAML PREREQUISITES	20
Shibboleth Source – http://shibboleth.internet2.edu	20
libcurl – http://curl.haxx.se/libcurl	21
Log4cpp 3.5RC1	21
Xerces-c 2.6.1 – http://xml.apache.org/xerces-c/	21
XML-Security-c 1.1 – http://xml.apache.org/security/c/	21
INSTALLING OPENSAML 1.0	22
INSTALLING SHIBBOLETH 1.2.1A.....	22
EDIT CONFIGURATION FILES.....	23
httpd.conf	23
Shibboleth.xml.....	23
SETTING UP THE ATTRIBUTE ACCEPTANCE POLICY	24
PROTECTING WEBSITES IN APACHE.....	25
USEFUL LOG FILES	25
CREATING A SELF SIGNING CERTIFICATE AUTHORITY	26
HOW DO I SIGN A CSR FILE?	26
CREATING YOUR OWN WAYF.....	27

Introduction

Purpose

The purpose of this document is to reduce the steep learning curve that is associated with Shibboleth and its associated technologies, the document is broken down into sections, firstly an overview of Shibboleth is given, this section is not highly technical and is aimed to be read by management staff who may not actually be implementing the technology itself but would still like an understanding of how Shibboleth works. Following this section is the actual installation and configuration of both the IdP and SP components.

The document also provides information on setting up your own WAYF and certification authority, this may be useful for institutions who wish to create their own federations, either for internal use, or for a small limited set of sites.

Note that this document refers to version 1.2 of the Shibboleth implementation.

Prerequisites

The install sections within this guide are based on Fedora Core 4, at least a basic understanding of Linux is therefore required to complete the installation parts of this document. If you are using a different operating system there may be differences in the commands and file locations used, however the general installation instructions given will provide guidance regardless of the environment used.

Shibboleth Overview

What is it?

Shibboleth is an open source system, developed by Internet2. It allows institutions to share resources with each other by providing a mechanism which sits between those sites concerned, this is known as "middleware". It was designed to address the issues currently seen in existing technologies used to share resources, such as access based on IP addresses, access through proxy servers and access by duplicating user accounts at each institution.

Shibboleth provides a framework that allows a user to authenticate against his own user database to access a web resource on another institutions domain, regardless of his location. Users are not logged onto the resource itself, instead, attributes for the user are sent to the resource from the home institutions database, these attributes are then used to decide the type of access the user has on the site. For example, a student authenticates against their own user database to gain access to a repository at another university, specific attributes for the user will be released to the repository, in this case it may just be the course the student is on, or simply the institution they are from, the repository will then use these values (and not the initial logon credentials) for any authorisation decisions that need to be made.

The benefits of Shibboleth are many;

- Enables institutions to share resources easily between each other.
- Administrative overhead is reduced, as users do not need to be replicated and maintained at each site.
- Security is enhanced because only one account exists for all resources.
- User ID's do not have to be used to access a resource, thus privacy is maintained.

How does it work?

Shibboleth has two fundamental components, these being the Origin (Identity Provider) and the Target (Service Provider). Every machine with a web resource which is secured by Shibboleth is a target, and every organisation which authenticates users against their own database has an origin. Another component is also used with Shibboleth, though it can be bypassed in certain situations, this is the WAYF service (Where Are You From?).

A brief description of each component follows:

Origin (Identity Provider)

The origin comprises of four services: the HS (Handle Service), the AA (Attribute Authority), a directory service (user database) and a sign-on system (SSO).

HS

The Handle Service is provided by Shibboleth in the form of a web service written in JAVA. Its job is to give the end user a handle to be used later to request attributes from the AA. It does this by first determining whether the user has already logged on by consulting the SSO, if they have not they will be authenticated. The Handle will be produced and passed to the targets SHIRE.

AA

The AA service is also provided by Shibboleth, again, written in JAVA and run as a web service. The AA responds to requests from the targets SHAR service, the target will send the handle sent by the HS, this is then used to look for policies which say which attributes can be released, based on these policies the AA will then query the directory service for the attributes of the user, these are then sent to the targets SHAR.

Directory Service

The directory service is the organisations user database, normally in the form of an LDAP directory, for example iPlanet, Active Directory etc

Sign-On System

The sign-on system provides authentication for the user, this is separate from Shibboleth, but is triggered by the HS component of the origin when a user has not yet logged on. This trigger is made using standard web server authentication methods, for example, triggering the HS to get you authenticated could be done by protecting the HS with an ldap auth module, or a WebISO (Web Initial Sign-on) system such as Pubcookie.

Target (Service Provider)

The target consists of three components, the SHIRE (Shibboleth Indexical Reference Establisher), the SHAR (Shibboleth Attribute Requester), and the RM (Resource Manager). All of these components are part of the target implementation of Shibboleth, written in C.

SHIRE

The SHIRE contacts the origins HS, once the user has been authenticated via the HS, a handle will be sent back to the SHIRE. When the SHIRE has the handle, it will send it to the SHAR.

SHAR

The SHAR will use the handle given to it by the SHIRE along with the address of the AA to request attributes about the user, once these attributes have been passed to the SHAR, AAP's (Attribute Acceptance Policy) will be used to provide validation and analysis before passing the attributes to the RM.

RM

The Resource Manager is responsible for making access decisions based on the attributes sent to it by the SHAR.

WAYF (Where Are You From?)

In a federation where many organisations are present, the target must know which origin to get a handle and attributes from. The WAYF provides a mechanism for allowing users to be forwarded to the correct home organisation, this is normally presented in the form of a web page with a drop down list, once the correct organisation is selected, the user will be forwarded to their origin server ready for authentication.

Message Flow

The components explained above can be seen in Figure 1 below, to help you understand how these fit together and see the flow of the messages in a typical exchange we will go through the steps, labelled 1 to 10 on the diagram. We will continue the example given earlier and assume the target is a repository protected by Shibboleth, and that the origin is a university.

Shibboleth Architecture Overview

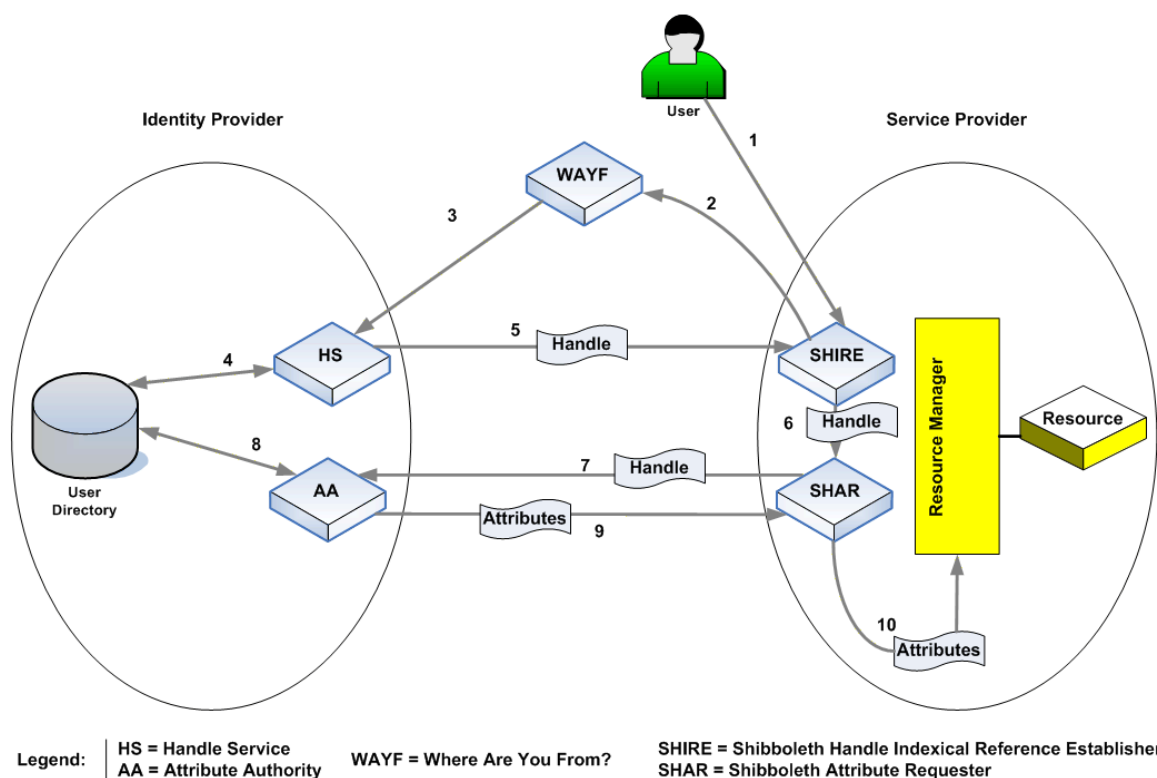


Figure 1 – Diagram of Shibboleth components and message flow

1. The student types the address of the repository in his web browser.
2. The SHIRE steps in and looks at its metadata to decide where to send the user, in this case it is the federations WAYF service. The student's browser will now be redirected to the WAYF, a web page with a list of all the institutions in the same federation. He selects his home university.
3. The WAYF redirects the users web browser to the origins HS.
4. The Handle Service of the origin will consult the origins sign-on system to see whether the student has logged on before, it has not, so the student is presented with a logon screen. He enters the same username and password which he uses when logging on locally at his university.
Note: The next steps are transparent to the student.
5. Once authenticated, the HS generates a handle, this is sent in a SAML authentication assertion to the targets SHIRE.
6. The SHIRE hands the handle to the SHAR.

-
7. The SHAR uses the handle with the address of the origins AA to request attributes about the student.
 8. The AA at the origin then uses the handle to consult the Attribute Release Policies set for the student, these tell the AA what attributes can be sent to the target. The attributes allowed to be released are queried against the origins directory service. In this case it is simply the users organisation that is released.
 9. The attributes are then sent to the SHAR on the target. Some basic validation and analysis is made based on the Attribute Acceptance Policies in place, these say what attributes the SHAR can accept and pass on for use in the web application.
 10. These attributes are then sent off to the resource manager, decisions on access are then made based on the values of these attributes. The student's organisation attribute is used to show that the student can have access to the repository. Note that the username and password are not used to make any authorisation decisions.

Federations

What is a federation?

A federation in Shibboleth is the collective term for a trusted group of Identity Providers and Service providers, the federation defines the policies that each member must adhere to, for example an IdP may have to use a certain certificate for it to be allowed into the federation for security reasons, attributes required to be released are also defined, if these rules are not met then the institution is not trusted so cannot be part of the federation, the federation therefore requires administration staff to write these policies and take care of the registration duties.

When accessing resources the list of IdP's you can authenticate against are those of your own federation, you can see who has an IdP in your federation every time you are sent to your WAYF, this is displayed in the drop down list, the federation owner will also show the SP's available and the attributes each one requires for the site to work correctly.

For further information on federations see the excellent guides at the SDSS development federation web site <http://sdss.ac.uk/>. Production federations are already in use in Switzerland <http://www.switch.ch/aai/tech/>, and Finland <http://www.csc.fi/suomi/funet/middleware/english/index.phtml>.

Identity Provider Implementation (v1.2.1)

Operating System

As mentioned in the introduction, this guide is written for Fedora Core 4, during the OS install process we only selected "Web Server" and "Development Tools".

Update process

We will be using yum to install and update some of the packages needed, to speed up the download process we will update the yum repository files to connect to a UK academic mirror instead (see <http://fedora.redhat.com/download/mirrors.html> to find a mirror that may be more appropriate to you).

```
# vi /etc/yum.repos.d/fedora.repo
```

Comment out the mirrorlist item, and add a new baseurl, so it looks like below.

```
[base]
name=Fedora Core $releasever - $basearch - Base
#baseurl=http://download.fedora.redhat.com/pub/fedora/linux/core/$releasever
/$basearch/os/
#mirrorlist=http://fedora.redhat.com/download/mirrors/fedora-core-
$releasever
baseurl=http://www.mirror.ac.uk/sites/fedora.redhat.com/$releasever/$basearc
h/os
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora
```

Now we will update the update repository.

```
# vi /etc/yum.repos.d/fedora-updates.repo
```

Comment out the mirrorlist item, and add a new baseurl, so it looks like below.

```
[updates-released]
name=Fedora Core $releasever - $basearch - Released Updates
#baseurl=http://download.fedora.redhat.com/pub/fedora/linux/core/updates/$re
leasever/$basearch/
#mirrorlist=http://fedora.redhat.com/download/mirrors/updates-released-
fc$releasever
baseurl=http://www.mirror.ac.uk/sites/fedora.redhat.com/updates/$releasever/
$basearch
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora
```

NTP configuration

If you have installed only the selected items as we did (see above), then NTP will need to be installed, if you have NTP installed you can skip this command I go straight to editing ntp.conf.

```
# yum install ntp
# vi /etc/ntp.conf
```

Comment out the existing server references in the "our timeservers" and insert your NTP server in the format "server <address of NTP server>".

Then to get xntpd to load automatically, even after a reboot, run the following commands.

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

Verify the NTP status.

```
# ntpq -p
```

Firewall configuration

Certain ports are required by Shibboleth, port 80 is not really needed, but can be useful for debugging in the setup stage, the same is true for Tomcat's port 8080 (these can be removed once you have Shibboleth up and running). Port 443 will be assigned to the Handle Service and port 8443 is required due to a bug in Apache 2, this will host the Attribute Authority.

Configure the firewall to allow ports 80 ,8080 ,443 and 8443, in Fedora the easiest way to do this is by running lokkit.

```
# lokkit
```

Select "Enabled" on the security level, then select "Customize", enter the port numbers in the "Other ports" section, using a comma to separate each one. Once done, click ok, and ok again on the main menu.

To verify the rules are now in iptables, we can list them.

```
# iptables -L
```

Install Apache and associated files

Now update Apache, we will also need the Apache development package as well, in turn, this will download apr, apr-devel, apr-util-devel, pcre and pcre-devel.

```
#yum install httpd
#yum install httpd-devel
```

Restart Apache to verify it is working correctly.

```
# service httpd restart
```

View <http://localhost> in your browser, a "Fedora Core Test Page" should appear.

To ensure Apache starts up automatically on a reboot run the following command.

```
# chkconfig httpd on
```

If you have decided to leave SELINUX installed we will need to install a package to allow us to add additional allowable actions, on an Identity Provider it will be the JK2 connector that requires this.

```
# yum install selinux-policy-targeted-sources
```

Install Java

Java is a core component of a Shibboleth origin. We will download and install the latest version of Java 5, at the time of writing this, Java 5 update 5 was the latest. It is recommended that you go to <http://java.sun.com> to ensure you are using the latest version.

```
# cd /usr/local/src
# wget http://wayf.kidderminster.ac.uk/shibfiles/jdk-1_5_0_05-linux-i586.rpm
# rpm -ivh jdk-1_5_0_05-linux-i586.rpm
# export JAVA_HOME=/usr/java/jdk1.5.0_05
```

Now add the last export line to /etc/profile to make it take effect on a reboot.

```
# vi /etc/profile
```

Insert the export command after any other export commands in the file, save and exit.

Install Tomcat 5

Download and install the latest version of Tomcat from <http://jakarta.apache.org/tomcat>, at the time of writing this guide, this is version 5.5.9.

```
# cd /usr/local/src
# wget http://www.mirrorservice.org/sites/ftp.apache.org/jakarta/tomcat-5/v5.5.9/bin/jakarta-tomcat-5.5.9.tar.gz
# tar -zxvf jakarta-tomcat-5.5.9.tar.gz
# ln -s jakarta-tomcat-5.5.9 tomcat
# export CATALINA_HOME=/usr/local/src/tomcat
```

Now add the above export line to /etc/profile to make it take effect on a reboot.

```
# vi /etc/profile
```

Insert the export command after any other export commands in the file, save and exit.

To make Tomcat start up automatically on a reboot we will create a script in init.d, note that the script downloaded here has a hard coded link to the Java install directory, if you are not using Update 5 or have installed it in another directory this will need to be updated.

```
# cd /etc/init.d
# wget http://wayf.kidderminster.ac.uk/shibfiles/tomcat
# chmod a+x tomcat
# chkconfig --add tomcat
```

Start up Tomcat.

```
# service tomcat start
```

View <http://localhost:8080> in your browser to verify that Tomcat is working correctly.

Install and setup jk2 connector

To allow apache to communicate with Tomcat, we need to install the JK2 connector.

```
# cd /usr/local/src
# wget http://apache.root.lu/jakarta/tomcat-connectors/jk2/source/jakarta-
tomcat-connectors-jk2-2.0.4-src.tar.gz
```

Stop Tomcat and Apache, otherwise changes made here may be overwritten.

```
# service tomcat stop
# service httpd stop
```

Extract the source and configure.

```
# tar -zxvf jakarta-tomcat-connectors-jk2-2.0.4-src.tar.gz
# cd jakarta-tomcat-connectors-jk2-2.0.4-src/jk/native2
# ./configure --with-apxs2=/usr/sbin/apxs \
--with-tomcat-4l=$CATALINA_HOME \
--with-apr-lib=/usr/lib \
--with-java-home=$JAVA_HOME \
--with-jni
```

Compile.

```
# make
# libtool --finish
# cd ../build/jk2/apache2
# apxs -n jk2 -i mod_jk2.so
```

Copy files to httpd modules, if asked to overwrite select yes.

```
# cp libjkjni.so /usr/lib/httpd/modules/jkjni.so
# cp mod_jk2.so /usr/lib/httpd/modules/mod_jk2.so
```

Edit your httpd.conf file.

```
# vi /etc/httpd/conf/httpd.conf
```

Tell apache to load the jk2 module by adding the following line to the Load Module block in your Apache httpd.conf file.

```
LoadModule jk2_module modules/mod_jk2.so
```

Save and exit.

If user authentication will make use of the REMOTE_USER value, then open the server.xml file;

```
# vi $CATALINA_HOME/conf/server.xml
```

And modify the *Connector port="8009"* section by adding *request.tomcatAuthentication="false"*, so it looks like below.

```
<Connector port="8009"
  enableLookups="false" redirectPort="8443" protocol="AJP/1.3"
  request.tomcatAuthentication="false"/>
```

Create a workers file.

```
# cd /etc/httpd/conf
# wget http://wayf.kidderminster.ac.uk/shibfiles/workers2.properties
```

Now start up Tomcat and Apache to test the new settings.

```
# service tomcat start
# service httpd start
```

Go to <https://localhost/jsp-examples> in your browser. This will test mod_ssl, Apache, Tomcat and the JK2 connector are all working together. If you have SELINUX enabled this will not work, as the JK2 connector is not allowed to run. Run the following commands to allow JK2 to work under SELINUX.

```
# cd /etc/selinux/targeted/src/policy
# audit2allow -i /var/log/audit/audit.log -l >> domains/misc/local.te
# make reload
```

Install Shibboleth

Download the Shibboleth Identity Provider tarball using one of the following commands:

```
# cd /opt
# wget http://wayf.internet2.edu/shibboleth/archive/shibboleth-origin-1.2.1.tar.gz
```

Expand the archive into a shibboleth-origin-1.2.1/ directory (/opt/ recommended):

```
# tar -zxvf shibboleth-origin-1.2.1.tar.gz
```

Move the Java .war file into Tomcat's tree:

```
# cp /opt/shibboleth-origin-1.2.1/dist/shibboleth.war
/usr/local/src/tomcat/webapps/
```

Restart Tomcat, which will automatically detect that there has been a new .war file added. This file will by default be expanded into /usr/local/src/tomcat/webapps/shibboleth.

```
# service tomcat restart
```

Create Certificates

The origin will need certificates to work correctly, here we will create a key and a certificate signing request, note that when asked the question for "Common Name", you must enter the fully qualified name of the server the certificate is to be used on.

```
# cd /etc/pki/tls
# openssl genrsa -out <thisservername>.key 2048
# openssl req -new -key <thisservername>.key -out <thisservername>.csr
```

You must now send the csr file to be signed by a CA, this could either be a third party CA such as Verisign, or could be your own internal CA.

If you currently have no Certificate Authority, and do not wish to purchase a certificate through a third party you can create your own CA thus allowing you to sign your own certificates. To do this, please see the section "Creating a self signing Certificate Authority".

Once you have received your certificate, copy the crt and key files into their respective directories.

```
# mv <thisservername>.key /etc/pki/tls/private
# mv <thisservername>.crt /etc/pki/tls/certs
```

If you are using a self signed CA you will want to download the CA root certificate, this will be used in the next section, if however you used a third party CA you will find it is already present in the ca bundle provided with Apache.

Configure Apache configuration files

We must now define a new SSL vhost in ssl.conf.

```
# wget http://rolo-kc-origin.kidderminster.ac.uk/shib/newvhost
# vi /etc/httpd/conf.d/ssl.conf
```

Scroll to the bottom of this file and type the following command in vi, this will insert the text file we have just downloaded;

```
:r newvhost
```

We must now point the SSLCertificateFile, SSLCertificateKeyFile (and SSLCACertificateFile if you self signed your cert) directives to the certificate and private keys generated in the previous step for each virtual host (443, and the newly inserted 8443). For example:

```
SSLCertificateFile /etc/pki/tls/certs /kc-origin3.crt
SSLCertificateKeyFile /etc/pki/tls/private/kc-origin3.key
SSLCACertificateFile /etc/pki/tls/certs /ca.crt
```

Above the virtual host we've just added is the default virtual host on port 443, we now need to add the handle service here. Insert the text below into the first virtual host just before the `</VirtualHost>`.

```
<Location /shibboleth/HS>
  AuthType Basic
  AuthName "My Shib Origin"
  AuthUserFile /etc/httpd/conf/user.db
  require valid-user
</Location>
```

Save and exit the `ssl.conf` file you've just been working on. We now need to add a couple of test users to the user database we are using to protect our handle service with.

```
# htpasswd -c /etc/httpd/conf/user.db testuser1
# htpasswd /etc/httpd/conf/user.db testuser2
```

Remove the `newvhost` file we just downloaded.

```
# rm newvhost
```

Restart Apache to make the changes take effect.

```
# service httpd restart
```

To test if this is all working ok, point your browser to <https://localhost/shibboleth/HS>, you should be prompted for a username and password, enter a user/pass that you just created, if successful you will see a redirect page which then goes to a page displaying "Handle Service Failure".

Edit Origin.xml

We now need to edit the `origin.xml` file, this is the main configuration file for the origin server, in it we will state the name of the server, the address for the Attribute Authority and the id for the server to be used in the federation, as well as other features such as logging and signing.

```
# vi $CATALINA_HOME/webapps/shibboleth/WEB-INF/classes/conf/origin.xml
```

Edit the following:

AAUrl should be changed to `https://<FQDN of this server>:8443/shibboleth/AA`

defaultRelyingParty is dependant on your federation data, if you have a sites file already set this to the SiteGroup Name, if it is the first, just enter your ProviderId.

providerId is either a URN or a URI, if you are joining a federation which you have no control over this will be assigned to you, otherwise this would normally be `https://<FQDN of this server>/shibboleth/origin`.

Uncomment the second part of the logging section so it looks like:

```
<!--
  <Logging>
    <Log4JConfig location="file:///tmp/log4j.properties" />
  </Logging>
-->

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

```
<TransactionLog location="file:///tmp/shib-access.log" />
</Logging>
```

Edit the Fileresolver Key and Certificate element paths (in the Credentials element) to point to your keys created earlier.

```
<Credentials xmlns="urn:mace:shibboleth:credentials:1.0">
  <FileResolver Id="foo">
    <Key format="PEM">
      <Path>file:///etc/pki/tls/private/<thisservername>.key</Path>
    </Key>
    <Certificate format="PEM">
      <Path>file:///etc/pki/tls/certs/<thisservername>.cert</Path>
    </Certificate>
  </FileResolver>
</Credentials>
```

Update FederationProvider uri to a suitable path. E.g. rolo-sites.xml.

```
<FederationProvider
type="edu.internet2.middleware.shibboleth.metadata.provider.XMLMetadataLoadW
rapper" uri="/conf/kidder-sites.xml"/>
```

Save and Exit.

Edit sites.xml

If you are responsible for your federations sites metadata you will now need to update it with your new origin site data.

Open up the sites file you specified in the FederationProvider tag in your origin.xml file. If this is your first origin, you can just edit the sites.xml file in the tomcat directory. If however you already have another origin setup, you may wish to update that one, and pull it down to this server (please see Federations section for more information on this).

```
# vi $CATALINA_HOME/webapps/shibboleth/WEB-INF/classes/conf/sites.xml
```

All of the servers in the federation must be defined in this file, the format is shown below, items in bold must be changed to reflect your new install. Note that the name for OriginSite must match your ProviderId as specified in your origin.xml file.

```
<OriginSite Name="https://FQDN of this server/shibboleth/origin">
  <Alias>Alias for Site</Alias>
  <Contact Type="technical" Name="contact name" Email="contact@yoursite" />
  <HandleService Location="https://FQDN of this server/shibboleth/HS"
Name="FQDN of this server" />
  <AttributeAuthority Location="https://FQDN of this
server:8443/shibboleth/AA" Name="FQDN of this server" />
  <Domain>Domain Name String here</Domain>
</OriginSite>
```

Setting up the Attribute Resolver

We now need to setup Shibboleth so it can retrieve attributes from your store, in this example we will assume the attributes are in an LDAP server. Firstly we need to edit origin.xml again to edit the resolverConfig entry, this tells Shibboleth which file to look at for the attribute retrieval settings.

```
# cd $CATALINA_HOME/webapps/shibboleth/WEB-INF/classes/conf
# vi origin.xml
```

There are a few examples of attribute files in this directory, resolver.xml is the standard echo resolver, resolver.jdbc.xml provides a JDBC connector, and resolver.ldap.xml, this provides sample attributes and connectors for common LDAP servers.

```
<?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://rolo-kc-
origin.kidderminster.ac.uk:8443/shibboleth/AA"
  resolverConfig="/conf/resolver.ldap.xml"
```

Open resolver.ldap.xml.

```
# vi resolver.ldap.xml
```

At the top of this file are definitions of the attributes you want to retrieve, by default eduperson attributes are defined, it is likely that you have not yet extended your LDAP schema to contain these, standard LDAP attributes are contained below, uncomment these for now.

If you look at one of the attribute definitions you will see a line that says **DataConnectorDependency requires="directory"**, this refers to a JNDIDirectoryDataConnector element at the bottom of the file, this is where the actual LDAP server information is set, including server name, base DN for searching and logon credentials. The sample data connector shown below is used to connect to Microsoft Active Directory, note that an additional item has been added, java.naming.referral is necessary with some AD configurations to follow referrals.

```
<JNDIDirectoryDataConnector id="directory">
  <Search filter="sAMAccountName=%PRINCIPAL%">
    <Controls searchScope="SUBTREE_SCOPE" returningObjects="false" />
  </Search>

  <Property name="java.naming.factory.initial"
value="com.sun.jndi.ldap.LdapCtxFactory" />

  <Property name="java.naming.provider.url"
value="ldap://172.16.0.3:389/dc=kiddercoll,DC=local" />

  <Property name="java.naming.security.principal"
value="CN=yourAuthUser,CN=Users,DC=kiddercoll,DC=local" />

  <Property name="java.naming.security.credentials" value="xxxxxxxxxxxx" />

  <Property name="java.naming.referral" value="follow" />
</JNDIDirectoryDataConnector>
```

Setting up the Attribute Release Policy

We now have a Shibboleth Identity Provider which can retrieve attributes from our store, however, we must explicitly say which attributes we actually want to be released, we do this by editing the Attribute Release Policy (ARP) files, there is a site wide ARP file, and a user specific file. We'll edit the site file.

```
# cd $CATALINA_HOME/webapps/shibboleth/WEB-INF/classes/conf
# vi arps/arp.site.xml
```

The file contains rules specifying which targets Shibboleth will release particular attributes to, a simple ARP file is shown below, this will allow these LDAP attributes to be released to all targets.

```
<Rule>
  <Target>
    <AnyTarget/>
  </Target>

  <Attribute name="urn:mace:dir:attribute-def:userPrincipalName">
    <AnyValue release="permit"/>
  </Attribute>

  <Attribute name="department">
    <AnyValue release="permit"/>
  </Attribute>

  <Attribute name="urn:mace:dir:attribute-def:sn">
    <AnyValue release="permit"/>
  </Attribute>

  <Attribute name="urn:mace:dir:attribute-def:givenName">
    <AnyValue release="permit"/>
  </Attribute>

  <Attribute name="urn:mace:dir:attribute-def:mail">
    <AnyValue release="permit"/>
  </Attribute>

  <Attribute name="urn:mace:dir:attribute-def:title">
    <AnyValue release="permit"/>
  </Attribute>

  <Attribute name="urn:mace:dir:attribute-def:cn">
    <AnyValue release="permit"/>
    <!--<Value release="deny">test1</Value>-->
  </Attribute>

  <Attribute name="urn:mace:dir:attribute-def:description">
    <AnyValue release="permit"/>
  </Attribute>

  <Attribute name="urn:mace:dir:attribute-def:telephoneNumber">
    <AnyValue release="permit"/>
  </Attribute>

</Rule>
```

Useful log files

Tomcat log files:

`$CATALINA_HOME/logs/localhost.log`

Shibboleth Log files:

`/tmp/shib-error.log`

`/tmp/shib-access.log`

Apache log files:

`/var/log/httpd/error_log`

`/var/log/httpd/ssl_error_log`

Service Provider Implementation (v1.2.1a)

Install Web server

The Shibboleth Target is an Apache dynamic loadable module. Therefore, it must be linked against Apache server and requires the Apache's APXS tool and Apache header files. In Fedora you will need to install the following rpm files to achieve this.

To speed up the update process, we'll update the Redhat network source file so we connect to a UK academic server, we have already modified this file, to download and use this file run these commands. If you would like to use different mirrors just edit the existing sources file.

```
#cd /etc/sysconfig/rhn/  
#mv sources sources.bak  
#wget http://wayf.kidderminster.ac.uk/shibfiles/sources
```

Now update Apache, we will also need the Apache development package as well, in turn, this will download apr, apr-devel, apr-util-devel, pcre and pcre-devel.

```
#yum install httpd  
#yum install httpd-devel
```

Restart Apache to verify it is working correctly.

```
# service httpd restart
```

View <http://localhost> in your browser, a "Fedora Core Test Page" should appear.

If you have decided to leave SELINUX installed we will need to install a package to allow us to add additional allowable actions, on a Service Provider it will be the Shibboleth module that requires this.

```
# yum install selinux-policy-targeted-sources
```

Installing OpenSAML Prerequisites

Shibboleth Source – <http://shibboleth.internet2.edu>

The components are to be built and installed to /opt/shibboleth-1.2 to avoid interference from preinstalled, older versions of libraries. Here we are just extracting the source code, we will compile and install it later.

Download the Shibboleth target 1.2 tarball.

Extract to /opt/shibboleth-1.2.1

```
# cd /opt  
# wget http://wayf.internet2.edu/shibboleth/archive/shibboleth-1.2.1a.tar.gz  
# tar -zxvf shibboleth-1.2.1a.tar.gz
```

libcurl – <http://curl.haxx.se/libcurl>

Extract tarball

```
# cd /usr/local/src
# wget http://curl.haxx.se/download/curl-7.12.2.tar.gz
# tar -zxvf curl-7.12.2.tar.gz
```

Install Curl

```
# cd curl-7.12.2
# ./configure --without-ca-bundle --enable-static=no --with-ssl=/usr/include/openssl \
--prefix=/opt/shibboleth-1.2.1
# make
# make install
```

Log4cpp 3.5RC1

```
# cd /usr/local/src
# wget http://wayf.internet2.edu/shibboleth/archive/log4cpp-0.3.5rc1.tar.gz
# tar -zxvf log4cpp-0.3.4.b.tar.gz
```

Install Log4cpp

```
# cd log4cpp-0.3.5rc1
# ./configure --prefix=/opt/shibboleth-1.2.1 --with-pthreads=yes --enable-static=no --enable-doxygen=no CXXFLAGS="-pthread"
# make
# make install
```

Xerces-c 2.6.1 – <http://xml.apache.org/xerces-c/>

Extract tar ball

```
# cd /usr/local/src
# wget http://wayf.internet2.edu/shibboleth/archive/xerces-c-src_2_6_1.tar.gz
# tar -zxvf xerces-c-src_2_6_1.tar.gz
```

Install Xerces

```
# export XERCESCROOT=/usr/local/src/xerces-c-src_2_6_1
# cd $XERCESCROOT/src/xercesc
# ./runConfigure -P/opt/shibboleth-1.2.1 -plinux -gcc -xg++ -minmem -nsocket -tnative -rpthread
# make
# make install
```

XML-Security-c 1.1 – <http://xml.apache.org/security/c/>

Extract tar ball

```
# cd /usr/local/src
```

```
# wget http://wayf.internet2.edu/shibboleth/archive/xml-security-c-1.1.0.tar.gz
# tar -zxf xml-security-c-1.1.0.tar.gz
```

Install XML-Security

```
# export XSECCROOT=/usr/local/src/xml-security-c-1.1.0
# cd $XSECCROOT/src
# ./configure --prefix=/opt/shibboleth-1.2.1 --without-xalan
# make
# make install
```

Installing OpenSaml 1.0

OpenSAML 1.0.1 – <http://www.opensaml.org/>

OpenSAML is a nearly complete implementation of SAML 1.1 (<http://www.oasis-open.org/committees/security/>).

Extract OpenSAML

```
# cd /usr/local/src
# wget http://wayf.internet2.edu/shibboleth/opensaml-1.0.1.tar.gz
# tar -zxvf opensaml-1.0.1.tar.gz
# cd opensaml-1.0.1
```

Install OpenSAML

```
# ./configure --prefix=/opt/shibboleth-1.2.1 --with-xerces=$XERCESSCROOT --with-curl=/opt/shibboleth-1.2.1 --with-xml-sec=$XSECCROOT --with-log4cpp=/opt/shibboleth-1.2.1 -C
# make
# make install
```

Installing Shibboleth 1.2.1a

```
# cd /opt/shibboleth-1.2.1
# ./configure --prefix=/opt/shibboleth-1.2.1 --enable-apache-20 --with-apxs2=/usr/sbin/apxs --with-log4cpp=/opt/shibboleth-1.2.1
# make
# make install
```

Add /opt/shibboleth-1.2.1/lib to the LD_LIBRARY_PATH environment variable. To do this add the following line to /etc/profile.

```
export LD_LIBRARY_PATH=/opt/shibboleth-1.2.1/lib
```

```
# cd /etc/init.d
# wget http://wayf.kidderminster.ac.uk/shibfiles/shar
# chmod a+x shar
# chkconfig --add shar
# chkconfig shar on
```

Edit Configuration files

httpd.conf

```
# vi /etc/httpd/conf/httpd.conf
```

Go to bottom of file and type:

```
Include /opt/shibboleth-1.2.1/etc/shibboleth/apache2.config
```

Save and exit.

If you have SELINUX enabled this will not load the Shibboleth module, run the following commands to allow it to work under SELINUX.

```
# service httpd restart
# cd /etc/selinux/targeted/src/policy
# audit2allow -i /var/log/audit/audit.log -l >> domains/misc/local.te
# make reload
# service httpd restart
```

Note: You may have to do the above a number of times to trigger all of the attempts made by the module.

Shibboleth.xml

```
# vi /opt/shibboleth-1.2.1/etc/shibboleth/shibboleth.xml
```

1)

```
<SHIRE>
<RequestMapProvider>
<Host name="<FQDN of server>">
```

2)

```
<Applications providerId=https://<FQDN of server>/shibboleth/target>
```

3)

```
<Sessions wayfURL="<YOUR WAYF URL>">
```

4)

Comment out the inline <FederationProvider> element.

5)

```
<saml:Audience>https://<FQDN of server>/shibboleth/target</saml:Audience>
```

6)

```
<FileResolver Id="defcreds">
  <Key format="PEM">
    <Path>Path to your Key</Path>
  </Key>
  <Certificate format="PEM">
    <Path>Path to your Certificate</Path>
  </Certificate>
</FileResolver>
```

Turn on debugging.

```
# vi /opt/shibboleth-1.2.1/etc/shibboleth/shar.logger
```

Change the INFO on the first line to DEBUG, so it now looks like:

```
log4j.rootCategory=DEBUG, shar_log
```

Save & exit.

```
# vi /opt/shibboleth-1.2.1/etc/shibboleth/shire.logger
```

Change the INFO on the first line to DEBUG, so it now looks like:

```
log4j.rootCategory=DEBUG, shire_log
```

Save and exit.

Setting up the Attribute Acceptance Policy

Your Shibboleth Service Provider will not accept any old attributes, you must specify which ones to accept, the AAP.xml file allows you to set these. Each attribute entry can have a header and/or alias set, the header maps the attribute to a HTTP header, whilst setting the alias allows mapping of the attribute to a htaccess rule.

```
# cd /opt/shibboleth-1.2.1/etc/shibboleth
# vi AAP.xml
```

The following sample would map the CN and mail attributes to a HTTP header, allowing web applications to base authorisation decisions, in addition the CN attribute has an alias set, this means we can set authorisation settings in htaccess rules.

```
<AttributeRule Name="urn:mace:dir:attribute-def:cn" Header="Shib-Person-
commonName" Alias="commonName">
  <AnySite>
    <AnyValue/>
  </AnySite>
</AttributeRule>

<AttributeRule Name="urn:mace:dir:attribute-def:mail" Header="Shib-Person-
mail">
  <AnySite>
    <AnyValue/>
  </AnySite>
</AttributeRule>
```

Protecting Websites in Apache

Below is an example of how to protect a website (or part of a website) in Apache, the first two lines in the location directive state that we want Shibboleth to protect it, the third tells it we want all of the following conditions to be true for access to the resource to be granted. In this case the conditions for access are that the "commonName" must have a suffix of "@kidderminster.ac.uk", and they must have either "000KC4", "Staff" or "staff" in their department attribute.

Any number of conditions can be made here, in this example we are using just simple regular expressions, using more complex regular expressions can provide you with a powerful method for blocking access to your web resources.

```
<Location /cshb/auth/shibboleth/shib-protected.php>
  AuthType shibboleth
  ShibRequireSession On
  ShibRequireAll on
  require commonName ~ .@kidderminster.ac.uk
  require department ~ .000KC4. [sS]taff
</Location>
```

Useful log files

Shibboleth Logs:

```
/opt/shibboleth-1.2.1/var/log/shibboleth/shar.log
/opt/shibboleth-1.2.1/var/log/shibboleth/shire.log
/opt/shibboleth-1.2.1/var/log/shibboleth/transaction.log
```

Apache log files:

```
/var/log/httpd/error_log
/var/log/httpd/ssl_error_log
```

Creating a self signing Certificate Authority

Produce your CA certificate; you can use this to sign all your certificate requests for the targets and origin servers in your federation. This method is mainly used for testing purposes, and some federations will not allow the use of certificates unless they are signed by a particular CA.

```
# cd /usr/share/ssl
# vi openssl.conf
```

Edit dir = .

```
[ policy_match ]
countryName          = match
stateOrProvinceName = optional
organizationName     = optional
organizationalUnitName = optional
commonName           = supplied
emailAddress         = optional
```

```
# echo '100001' >serial
# touch index.txt
# openssl req -new -x509 -extensions v3_ca -keyout private/cakey.pem -out
cacert.pem -days 3650 -config ./openssl.cnf
```

How do I sign a CSR file?

```
# cd /usr/share/ssl
# openssl ca -out kvle.crt -days 365 -config ./openssl.cnf -infile kvle.
csr
```

A certificate will now be present in the directory, this can be emailed back to the sender for use on the server.

Meta data:

```
# openssl x509 -in cert.pem -subject -nameopt RFC2253,sep_comma_plus_space
```

Note:

If IIS is used you will need to convert to pkcs12 format to import. Run this command:

```
# openssl pkcs12 -export -in kc-repository.shib.crt -inkey kc-repository.key
-out kc-repository-IIS.p12
```

Creating your own WAYF

Compile WAYF code.

```
# cd /opt/shibboleth-origin-1.2.1
# ./ant dist-wayf
# cp dist/shibboleth-wayf.war $CATALINA_HOME/webapps
```

Edit ssl.conf

```
# vi /etc/httpd/conf.d/ssl.conf
```

Insert the following to the first Virtual Host.

```
<Location /shibboleth-wayf>
    SSLOptions +StdEnvVars
</Location>
```

Restart Apache and Tomcat (see origin section for tomcat script if it is not present).

```
# service httpd restart
# service tomcat restart
```

Copy the sites file from the shibboleth directory to the wayf webapps directory.

```
# cp $CATALINA_HOME/webapps/shibboleth/WEB-INF/classes/conf/sites.xml \
$CATALINA_HOME/webapps/shibboleth-wayf/WEB-INF/classes/conf/
```

The WAYF is now accessible at <https://localhost/shibboleth-wayf/WAYF>