EnLIGHTTened HARC Acceptor Installation Document

Version 1.9

EnLIGHTTened Project
CCT at LSU

Jon MacLaren

August 22, 2007
# Contents

## 1 Introduction

1.1 Structure of the Document ........................................... 4
1.2 Dependences .......................................................... 4

## 2 Planning the Installation

2.1 A Note on Java DNS Caching ........................................ 5
2.2 Planning the Acceptor Locations ................................... 6
2.3 Getting X509 Certificates for the Acceptors and Apache Instances ........................................ 6
2.4 Downloading the Code and Documentation from CVS .......... 7
2.5 Relational Database .................................................. 8
2.6 Other Software ........................................................ 8
   2.6.1 Installing and Configuring Java 1.5 ......................... 8
   2.6.2 Apache Ant .................................................... 8

## 3 Persistence Database

3.1 MySQL ............................................................... 9
3.2 Oracle ..................................................................... 10

## 4 Installing and Configuring Apache HTTP Server

4.1 Patching for GSI Support ............................................ 11
4.2 Building ............................................................... 11
   4.2.1 Issues on 64-bit Platforms .................................... 12
4.3 Configuring ........................................................... 12
   4.3.1 httpd.conf ..................................................... 12
   4.3.2 httpd-ssl.conf: Random Seed ............................. 13
   4.3.3 httpd-ssl.conf: Ports ....................................... 13
   4.3.4 httpd-ssl.conf: Server Name and Admin ................ 14
   4.3.5 httpd-ssl.conf: Server Certificate ......................... 14
   4.3.6 httpd-ssl.conf: Verify Client Certificates ............... 15
   4.3.7 httpd-ssl.conf: Passing and Logging Clients’ DNs ...... 16
4.4 Starting Apache and Testing the Configuration ............... 16
4.4.1 Checking from the Command Line ........................................... 16
4.4.2 Checking from a Web Browser ........................................... 17

5 Installing and Configuring Apache Tomcat ................................ 18
  5.1 Basic Configuration .................................................. 18
    5.1.1 Port Configuration ............................................ 18
    5.1.2 Tomcat Manager Login ...................................... 19
    5.1.3 Restarting Tomcat ............................................ 20
  5.2 Building mod_jk ...................................................... 20
  5.3 Linking Apache HTTPD and Tomcat with mod_jk ......................... 20
    5.3.1 Tomcat Configuration ...................................... 20
    5.3.2 Apache HTTPD Configuration ................................ 22
  5.4 Testing the Configuration ......................................... 22

6 Installing the Acceptors .................................................. 24
  6.1 Initial Tasks ...................................................... 24
    6.1.1 Installing the JDBC Driver .................................. 24
    6.1.2 Installing Tomcat’s Ant Jarfile ............................ 25
    6.1.3 Installing Tomcat’s Ant Jarfile ............................ 25
    6.1.4 Installing CA Certificates for Java ....................... 25
  6.2 Configuring .......................................................... 26
    6.2.1 Editing build.xml .......................................... 26
    6.2.2 Editing context.xml ........................................ 27
    6.2.3 Editing web.xml ............................................. 28
  6.3 Build and Deploy ................................................... 30
  6.4 Verifying the Installation ......................................... 30
1 Introduction

This document describes how to install the version of the HARC acceptors currently stored in the CCT CVS repository. I'll keep this up-to-date as the procedure changes.

This is part of the document set for Version 1.9, which has security, but which does not implement persistence correctly—this will be in Version 2.0. The APIs will not change between Version 1.9 and Version 2.0, except that support for plain HTTP will disappear.

1.1 Structure of the Document

Figure 1 shows the basic architecture of HARC. The key to installing HARC is the installation of a working set of acceptors. The next sections explain how to plan an installation of acceptors, and how to then deploy the acceptors to follow that plan.

1.2 Dependences

The Acceptors are written in Java 1.5. The only feature of 1.5 that was really required was the timeout on HTTP connects and POST/GETs. However, that is essential to HARC.

Table 1 summarises the components that are required for the installation to be carried out. The installation and configuration of each component is described in the instructions below.
2 Planning the Installation

2.1 A Note on Java DNS Caching

By default, Java JVM’s cache DNS lookups forever. As Acceptors live inside a Tomcat container, this means that DNS lookups are cached until the container is restarted; redeploying the Acceptor’s servlet has no effect. This creates some difficulties for the Acceptors, as sometimes DNS entries for head nodes of clusters are changed for legitimate reasons. This can cause the Acceptors to keep sending messages to the old IP address of a Resource Manager, long after the change has propagated through DNS.

Consider the following, specific example. Some clusters are configured to have a public DNS name, e.g. zeke.loni.org, which is an alias for the head node, e.g. 13f1n01.sys.loni.org. If the head node fails, and another node needs to be brought in to act as the head node, this alias would be changed to point to the new head node, e.g 13f1n03.sys.loni.org. The head node the node of a cluster acting as the head node might be changed, and the public DNS name of the cluster be changed to point to this new node. If the default behavior of the Java installation used by Tomcat is not changed, the Acceptor will not see this change, and clients will not be able to co-allocate this resource until the Tomcat container is restarted.

The settings for the Java DNS caching cannot be configured from the command line. Rather, they have to be set globally in the java.security file, which usually lives in:

```
$JAVA_HOME/jre/lib/security/java.security
```

The two properties in this file which are of interest are networkaddress.cache.ttl and networkaddress.cache.negative.ttl; these control the caching of successful and failed lookups respectively. They are set to the number of seconds to cache the results for, with the special values of 0 for “never cache”, and -1 for “cache forever”. If they are not set at all, networkaddress.cache.ttl defaults to -1 (cache forever), and networkaddress.cache.negative.ttl defaults to 10 seconds.¹

Inside the java.security file, when you search for the above properties, you will likely find the following warning above the setting for the successful-lookup value:

```
# NOTE: setting this to anything other than the default value can have
# serious security implications. Do not set it unless
# you are sure you are not exposed to DNS spoofing attack.
```

I believe that this advice is misguided. Java is an application of sorts, and it is simply not its place to try to “add security” in this way. As results are only cached for the lifetime of the JVM, (Java has no mechanism to cache the DNS settings in a machine-wide, or persistent fashion), this means that two Java programs which start on the same machine at different times could return different values for hostname lookups, and worse, would continue to do so until they terminated. This strikes me as pretty crazy. And given that it is typical for DNS servers to only keep DNS lookups for 24 hours, why should Java keep them for months on end?

Therefore, I recommend that you change the value of networkaddress.cache.ttl to 3600 (1 hour), or, if you want to be more conservative, to 86400 (24 hours)—but remember that this value would

¹The source for this information is the Java API documentation for the class java.net.InetAddress, e.g. http://java.sun.com/j2se/1.5.0/docs/api/java/net/InetAddress.html, and the API documentation on Networking Properties, e.g. http://java.sun.com/j2se/1.5.0/docs/guide/net/properties.html.
mean that your Acceptor would not see DNS changes for up to 24 hours after you can see them on the command line. (You will probably need to uncomment the line where this property is set.) No change to `networkaddress.cache.negative.ttl` is necessary.

The Acceptor servlet examines these properties on startup, and logs their values in the Tomcat log file (and gives a warning if it thinks the value is too long). This will allow you to check that your changes have been propagated. Remember that the changes to `java.security` will not take effect until you have restarted the Tomcat container.

2.2 Planning the Acceptor Locations

First, you need to plan where to deploy the acceptors. HARC keeps working as long as a majority of acceptors are working. There must be an odd number of acceptors, $N$, where $N > 1$. Acceptors can be kept on any machines—they do not need to be on or near the resources that they are co-allocating. The acceptors need to be on globally reachable machines, as clients will connect to them.

You want the acceptors to run on reliable machines, where possible. The machines do not need to be particularly fast. The acceptors should all be on different machines—otherwise the failure of a single machine might cause the failure of multiple acceptors. The machines should be distributed in such a way as to minimise the impact caused by a single LAN failure.

The locations might be something like:

https://glambda-dev.enlightenedcomputing.org:9877/currituck  
https://kite1.enlightenedcomputing.org:9877/bodie  
https://vclb5-1.hpc.ncsu.edu:9877/hatteras  
https://scoop.cct.lsu.edu:9877/ocracoke  
https://131.215.207.26:9877/lookout

Note that we use different URI endings for the different acceptors. This is not strictly necessary (i.e. we could replace currituck, bodie, etc. with ‘acceptor’), however it is a convenient way to identify the acceptors, and allows for the possibility that more than one acceptor might be running in the same Tomcat instance. When we come to configure the acceptors in Section 6.2, we shall use the term `acceptor name` to mean this suffix, i.e. currituck, bodie, etc.

The HARC Acceptors run as Tomcat servlets, with Apache HTTP Server as a front end. If you are installing the acceptor in a pre-existing Apache instance, this might govern which ports you use. Note that the single https port is sufficient to support authentication by both plain X509 certs, and GSI proxies.

To avoid delays due to DNS lookup problems, it is recommended to use the IP addresses instead of the hostnames inside the acceptor definitions. To make this document readable, however, we use hostnames.

2.3 Getting X509 Certificates for the Acceptors and Apache Instances

Each of the Acceptors will require a valid certificate, which it will use when posting messages to the other Acceptors and to HARC RMs. In addition, if you are setting up new Apache instances, then each
of these will also require a valid certificate. It is possible to use the same credential for both purposes, but not necessary. (For example, system administrators may not wish to provide the Acceptor access to the host certificate being used by an already existing Apache instance.)

The X509 Distinguished Name (DN) that you can choose for the Acceptor will depend upon the Signing Policy of the Certificate Authority you get the certificate from. In the Grid world, the certificate will often be the Globus-style "host certificate", with a DN like:

```
C=UK, O=eScience, OU=Manchester, L=MC,
   CN=deepsouth.mvc.mcc.ac.uk/emailAddress=jon.maclaren@manchester.ac.uk
```

However, you may be able to get something more descriptive, such as:

```
C=UK, O=eScience, OU=Manchester, L=MC,
   CN=harcacceptor/deepsouth.mvc.mcc.ac.uk/emailAddress=....
```

This latter example would not be appropriate for dual use as the Apache instance's credential.

**UK e-Science CA**  This CA now issues service certificates for the service name harcacceptor. These are recommended for use as HARC Acceptor Credentials.

### 2.4 Downloading the Code and Documentation from CVS

1. Go to the directory where you want to place the code, e.g.
   ```
   cd $HOME/cvs
   ```

2. Set your CVSROOT up.
   ```
   export CVSROOT=:pserver:cvs_anon@cvs.cct.lsu.edu:/public
   ```

3. Log in, entering "anon" when prompted for your CVS password.
   ```
   cvs login
   ```

4. Check out the code and documentation for V1.9.
   ```
   cvs co -r V1_9 negotiation
   ```
   A directory called `negotiation` will be created under the current directory, containing the Acceptor and RM sources, plus the documentation.

5. Set the environment variable `HARC` to point to the `negotiation` directory. Put this into your `.bash_profile`, etc. This variable will be referred to a lot in this and other documentation.
2.5 Relational Database

For the Acceptors to be correctly repairable, they must keep an up-to-date copy of their state in an RDBMS. Each acceptor has its own database, which must be hosted on the same node where the acceptor runs (in order to avoid creating three new points of failure for the acceptor; the database, the machine the database is hosted on, and the network connecting the database machine to the acceptor machine). When the acceptor restarts, the state is reloaded from the database. The state in the database is constantly updated, so that if—at any time—the acceptor (or the machine it is running on) is killed, it may be restarted and not lose state. This is called persistence, and we call the database the *persistence database*.

At present, there are instructions for MySQL, and also notes for Oracle. You should be able to use any Relational Database for which there is a JDBC Driver for. There is a page on the Tomcat page about how the database gets configured for use through Tomcat (as a JNDI Datasource). The page deals specifically with MySQL, Oracle and PostgreSQL. There are different versions for Tomcat 5.5.x and 6.0.x:

5.5 – http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html

If you do not have Oracle already available and installed, we recommend MySQL 5.0.x.

2.6 Other Software

2.6.1 Installing and Configuring Java 1.5

You must use Java 1.5 (sometimes called Java 5). You will need the JDK, not just the JRE. You need to point the environment variable `JAVA_HOME` to the location of the SDK, and add this to your `.bash_profile`, etc., e.g.:

```
export JAVA_HOME=/usr/java/jdk1.5.0_08
export PATH=$JAVA_HOME/bin:$PATH
```

2.6.2 Apache Ant

The version of Ant that is used is not too important. Versions 1.6.5 and 1.7.0 have both been tested. If you need to install this yourself, you should set `$ANT_HOME` in your `.bash_profile` and add `$ANT_HOME/bin` to your PATH. Ant can be downloaded from:

http://ant.apache.org/
3 Persistence Database

For the Acceptors to be correctly repairable, they must keep an up-to-date copy of their state in an RDBMS. Each acceptor has its own database, which must be hosted on the same node where the acceptor runs (in order to avoid creating three new points of failure for the acceptor; the database, the machine the database is hosted on, and the network connecting the database machine to the acceptor machine).

3.1 MySQL

Each acceptor needs a MySQL database. I’ve used MySQL 5.0.24a, and recommend that version. You should be able to find RPMs for this on http://www.mysql.com. The MySQL instance is used to store the internal state of the acceptor. As such, it should be installed

Once MySQL is installed on the node, you should create a database for the acceptor to connect to. You will use the name of the database which is created, and the login information later.

We recommend using acceptor_<name> for the database instances, using the name of the acceptors decided in Section 2.2. For the database username, we recommend using the account name that the acceptor will be running under, to access the database.

The database should be created after logging into the database. An example is shown here:

1. Login to mysql as root (when prompted for your password, pressing return will probably work, by default, i.e. unless you’ve set it already).

   mysql -p -u root

2. Create the database

   create database acceptor_currituck;

3. Create an account so that the acceptor can access the database, create tables, etc. Here I assume that the account being used to run the acceptor is called "harc". Please don’t use "pass" as the password for real.

   grant all privileges on acceptor_currituck.*
   to 'harc'@'localhost' identified by 'pass' with grant option;

   grant all privileges on acceptor_currituck.*
   to 'harc'@'localhost.localdomain'
   identified by 'pass' with grant option;

4. Note the database name, username and password down—these will be used again when we configure the acceptor.

5. Exit mysql (type "quit" then return)

6. Log back into the database using the new account, e.g.:

   mysql -p -u harc acceptor_currituck;
7. Create the tables. You will need to cut and paste the commands from $HARC/documentation/acceptors/mysql.txt into the mysql prompt.

3.2 Oracle
4 Installing and Configuring Apache HTTP Server

This section covers the basic setup of Apache. There is a separate section describing the linking of Apache to Tomcat.

We recommend using Apache 2.2.x. We have tested against 2.2.3, and also against 2.2.4.

You can download the HTTP Server from:

http://httpd.apache.org/download.cgi

There is excellent documentation on the HTTP Server, which this isn’t intended to replace, e.g.

http://httpd.apache.org/docs/2.2

Here we just deal with the minimal configuration items that need to be set for HARC. We don’t deal with how to optimize the installation, etc.

4.1 Patching for GSI Support

If you want to support GSI proxies in addition to plain X509 certificates, then you need to perform this extra step before you configure and build. Note that the following patch is specific to OpenSSL 0.9.7x, and does not build against 0.9.8x. However, at the time of writing, 0.9.7 is still being actively patched, with new releases still appearing. I’ve checked this patch with OpenSSL 0.9.7m.

After you have unpacked your Apache 2.2.3 or 2.2.4 build, then change directories to the top of the unpacked tarball, e.g.:

```
cd httpd-2.2.3
```

Then type:

```
patch -p0 < $HARC/sample/tools/apache/apache-2.2.x-GSI.patch
```

You should see the following output (and not be prompted):

```
patching file modules/ssl/ssl_private.h
patching file modules/ssl/ssl_engine_init.c
patching file modules/ssl/ssl_engine_kernel.c
```

This concludes the patching process.

4.2 Building

When building Apache, you should tell configure to compile in mod-ssl:
./configure --prefix=/usr/local/apache2 --enable-ssl

or, if you have a non-standard location for openssl:

./configure --prefix=/usr/local/apache2 --enable-ssl --with-ssl=/usr/local

Then:

make
sudo make install

You may well have other arguments to pass to configure.

4.2.1 Issues on 64-bit Platforms

At the time of writing, Apache HTTPD does not build correctly out-of-the-box on some 64-bit platforms, e.g. Fedora Core 6, due to a configuration problem. If you get the error:

/usr/lib/libexpat.so: could not read symbols: File in wrong format

when building, then you will need to manually edit a Makefile manually, after the configure, but before the make, as described on the following page from the Apache bugzilla:

http://issues.apache.org/bugzilla/show_bug.cgi?id=41308

The bug, a duplicate of bug 28205, was fixed on the 8th of January 2007. It should be fixed in versions of HTTPD following 2.2.4.

4.3 Configuring

In this section, we use $HTTPD to refer to the root of the Apache HTTP Server installation. In the configuration example from the previous section, this would point to /usr/local/apache2.

Most of this section deals with the configuration files httpd.conf and httpd-ssl.conf. The location of these might be a little different on some systems (particularly if you are using an Apache installation that came bundled with your machine). When building and installing it yourself, these should be at:

$HTTPD/conf/httpd.conf
$HTTPD/conf/extra/httpd-ssl.conf

4.3.1 httpd.conf

If you are only using the Apache instance for hosting the acceptor (or for the acceptor plus other secure services), you can comment out the line reading:
# Listen 80

Also, enable https by uncommenting the line:

```
Include conf/extra/httpd-ssl.conf
```

### 4.3.2 httpd-ssl.conf: Random Seed

Uncomment one pair of the SSLRandomSeed lines—for /dev/random or /dev/urandom. urandom is better, because it never blocks; random can block until there’s been enough entropy generated by the system. Please check that the one you want to use exists. You should end up with:

```
#SSLRandomSeed startup file:/dev/random 512
SSLRandomSeed startup file:/dev/urandom 512
#SSLRandomSeed connect file:/dev/random 512
SSLRandomSeed connect file:/dev/urandom 512
```

### 4.3.3 httpd-ssl.conf: Ports

If you are not using the default HTTPS port (443) for the Acceptor, then change the line:

```
Listen 443
```

to:

```
Listen 9877
```

If your machine has multiple IP addresses, you may want to make Apache listen only on a particular interface. To do this, you can specify an IP address in the Listen directive, e.g:

```
Listen 128.109.130.99:9877
```

Also, change the line:

```
<VirtualHost _default_:443>
```

to:

```
<VirtualHost _default_:9877>
```

**Firewall** Which ever port you are using, you should check your iptables settings. Make sure the port you are using is open.
4.3.4 httpd-ssl.conf: Server Name and Admin

Just under this, set the ServerAdmin line to your email address, and then set (and uncomment) ServerName to point to the host name and port, e.g.

```plaintext
ServerAdmin maclaren@cct.lsu.edu
ServerName scoop.cct.lsu.edu:9877
```

Use the IP address if you aren’t in DNS.

4.3.5 httpd-ssl.conf: Server Certificate

There are three files that you will need, to set up the Apache server’s certificate. The first two are the signed public certificate, the private key, both in PEM format. The third file should contain all the Certificate Authority public certificates, right up to the top-level self-signed CA; these must be in PEM format, and then concatenated together into a single file. As stated in Section 2.3, this may or may not be the same credential as used for the Acceptor. If your signed credential exists as a PKCS#12 bundle, you can extract the public and private parts as follows:

```plaintext
openssl pkcs12 -in bundle.p12 -clcerts -nokeys -out server.crt
openssl pkcs12 -in bundle.p12 -nocerts -nodes -out server.key
```

You will need to enter a password to unlock the PKCS#12 bundle.

These files are typically placed in `$HTTPD/conf`. The names and locations of the files are set in the httpd-ssl.conf file—see the definitions of SSLCertificateFile and SSLCertificateKeyFile and SSLCertificateChainFile. These will default to:

```plaintext
$HTTPD/conf/server.crt
$HTTPD/conf/server.key
$HTTPD/conf/server-ca.crt
```

Place the signed public certificate, private key, and CA certificates file in locations you chose. Please ensure that the file ACL for the private key is secure (e.g. using `chmod 400 server.key`) This last one will need to be uncommented.

Note that if you have a password on your key file, then you will be prompted for this when you start the server. This is usually undesireable, and so you can create a passwordless version key as follows:

```plaintext
mv server.key server.key.pass
openssl rsa -in server.key.pass -out server.key
chmod 400 server.key
```

Enter the password on the key when prompted to do so.
4.3.6 httpd-ssl.conf: Verify Client Certificates

We also need to configure Apache to verify the incoming connections too.

You can enforce very strict things on part of the tree served by Apache. But here we just configure Apache to authenticate the credentials presented by clients. The Acceptors will perform some authorization, only accepting Paxos messages from the other Acceptors; the RMs perform resource-specific authorizations.

To set this up, you need either a directory containing all the Certificate Authority (CA) public certificates, or a single file containing all of these. We only cover the directory-based set up, and advise that people use this method. You will need to have the CA Certificates used to sign the credentials of any HARC Acceptor or RM, any user of HARC, or any other Apache HTTPD instance housing the other Acceptors.

The directory should default to $HTTPD/conf/ssl.crt, but is set in the config file by the line containing: SSLCACertificatePath (which you may need to uncomment). You can either place the CA certificates (PEM format) into this directory using their hashed name (e.g. 5fb2fc80.0, etc.), or you can give them more meaningful names, e.g. cct-ca.crt—the suffix must be “.crt”, and create symbolic links to these from the hashed name. Some Apache distributions have a Makefile.crt that you can use to do this automatically; in this case, it will already be in the directory $HTTPD/conf/ssl.crt. In some Apache distributions, Makefile.crt no longer appears. We have placed a copy in the HARC CVS at $HARC/sample/tools/apache/Makefile.crt.

First, edit Makefile.crt, and put the full path of your OpenSSL program in the line that starts “SSL_PROGRAM=”, e.g.

    SSL_PROGRAM=/usr/bin/openssl

Then, in that directory, type:

    make -f Makefile.crt

This will generate symbolic links from the hashed names to the certificates. So you end up with something like:

    total 24
    lrwxr-xr-x 1 root wheel 10 Nov 29 15:24 5fb2fc80.0 -> cct-ca.crt
    -rw-r--r-- 1 root wheel 1538 Nov 29 15:24 Makefile.crt
    -rw-r--r-- 1 root wheel 912 Nov 29 14:57 cct-ca.crt

You should also keep an up-to-date Certificate Revocation List (CRL) for each of the CAs. This is a similar procedure. The directory for CRLs is set in the config file by the field SSLCARevocationPath (which will need to be uncommented), and defaults to $HTTPD/conf/ssl.crl. Again, you can place the files in the directory with the hashed names, e.g. 5fb2fc80.r0, or you can give them more meaningful names, e.g. cct-ca.crl—the suffix must be “.crl”, and create symbolic links to these from the hashed name. Some Apache distributions have a Makefile.crl that you can use to do this automatically; again, we’ve place a copy in $HARC/sample/tools/apache/Makefile.crt. The procedure for editing and using this Makefile is exactly the same as described above for the CA certificates.

Next, uncomment these two lines:
SSLVerifyClient require
SSLVerifyDepth 10

4.3.7 httpd-ssl.conf: Passing and Logging Clients’ DNs

There is a line in the default configuration file, commented out, defining the field SSLOptions. Instead of uncommenting this, add this line underneath it:

SSLOptions +StdEnvVars +ExportCertData

Finally, to log the Distinguished Name from the certificate in $HTTPD/logs/ssl_request_log change the CustomLog entry, so that it includes %{SSL_CLIENT_S_DN}x in the output format string, e.g.:

CustomLog /usr/local/apache2/logs/ssl_request_log \ 
"%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x %{SSL_CLIENT_S_DN}x "%r" %b"

4.4 Starting Apache and Testing the Configuration

Use the command: apachectl start to start the server (apachectl is located in $HTTPD/bin). The server should start, and will now present the correct certificate on the chosen port, e.g. 9877.

It is recommended that you verify that the web server is functioning correctly before you proceed. The following two sections show you how to do this, from either the command line, or from a browser. In both cases, you will need a personal X509 credential which has been signed by one of the CA Certificates that Apache has been configured to trust.

4.4.1 Checking from the Command Line

If your Apache instance is using a certificate where the Subject Name matches the hostname of the machine, then you can use the following curl commands to check that the server is setup correctly. If you have a different Subject Name, then the SSL verification in curl will fail; you can re-run the commands with an additional ‘-k’ flag.

    curl -v --cert $HOME/.globus/usercert.pem \ 
    --key $HOME/.globus/cct-userkey.pem \ 
    --capath /etc/grid-security/certificates \ 
    https://nodex-1.hpc.ncsu.edu:9877/

    If your key has a password, you will be prompted to enter it.

If you have built GSI support, you will also want to check using a GSI proxy, so now you should try:

    curl -v --cert /tmp/x509up_u'id -u' \ 
    --key /tmp/x509up_u'id -u' \ 
    --capath /etc/grid-security/certificates \ 
    https://nodex-1.hpc.ncsu.edu:9877/
In both cases, the output should end with something like:

```text
HTTP/1.1 200 OK
Date: Tue, 06 Mar 2007 14:39:49 GMT
Server: Apache/2.2.4 (Unix) mod_ssl/2.2.4 OpenSSL/0.9.7m
Last-Modified: Sat, 20 Nov 2004 20:16:24 GMT
ETag: "1674e9-2c-4c23b600"
Accept-Ranges: bytes
Content-Length: 44
Content-Type: text/html
* Connection #0 to host nodex-1.hpc.ncsu.edu left intact
* Closing connection #0
```

### 4.4.2 Checking from a Web Browser

If you want to check this with a Web Browser, then first try to connect to the document root of the web server, e.g:

```text
https://nodex-1.hpc.ncsu.edu/
```

without installing your certificate in the browser. You should not be able to connect. You might get a good error such as:

**Security Failure. Personal certificate required.**

But the error could be more cryptic (Firefox just gave me a bunch of numbers back).

If your certificate is not already in your browser, then you will need to install it. You should consult your browser documentation for instructions on how to do this. It’s likely that you will first need to create a PKCS#12 version of your credential. This contains your key, your cert, and the CA cert(s). To make this use:

```text
openssl pkcs12 -export -in usercert.pem -inkey userkey.pem
    -out jonmaclaren.p12 -name "Jon MacLaren" -certfile cct-ca.crt
```

You’ll get prompted for your key password, then a new password, which you’ll use when importing the cert into your browser.

You will now need to install the CA certificates as trusted certificates, then import the .p12 file you have just created.

Now try loading the web page, e.g. https://scoop.cct.lsu.edu:9877/ Try the page again—you should get some kind of default page. If you get "client certificate rejected", or similar, then you need to check the above configuration.
5 Installing and Configuring Apache Tomcat

The Acceptors have been validated against Tomcat 5.5 (using 5.5.20), using Java 1.5, and also against Tomcat 6.0.x (using 6.0.10) and Java 1.6.

You can obtain versions of Apache Tomcat from:

http://tomcat.apache.org/

Once you have your tarball, unpack it, and set CATALINA_HOME to point to the top directory. Add this to your .bash_profile, or similar. Also add $CATALINA_HOME/bin to your PATH. For example:

    export CATALINA_HOME=$HOME/apache-tomcat-5.5.20
    export PATH=$CATALINA_HOME/bin:$PATH

5.1 Basic Configuration

5.1.1 Port Configuration

When Tomcat starts up, it listens on a number of ports, which are configured in:

$CATALINA_HOME/conf/server.xml

The default version of this file defines the following ports:

- The shutdown port, which the server listens on for the shutdown command (default 8005);
- The plain HTTP port to listen for requests on (8080);
- The SSL HTTP port (8443, commented out);
- The AJP Connector port, used by mod_jk (8009);
- The Proxied HTTP port (8082, commented out).

All of these are defined in Connector elements, except for the shutdown port, which is defined by the port attribute of the single Server element (the root element of server.xml).

As we will be using the Apache HTTPD as a front end, this means that we only need to configure three ports:

1. The AJP Connector port, which receives requests from Apache HTTP;
2. The Plain HTTP port, which will be used to access the Tomcat manager to deploy the acceptor servlet; and
3. the shutdown port.
Note that none of these ports need to be accessible remotely. Apache HTTPD is the only thing that will talk to the AJP port, and shutdown.sh is the only thing that should be able to talk to the shutdown port. We will only use the Plain HTTP port to connect to the manager interface.

Note that if you have another Tomcat instance running on the machine, you must also make sure that the AJP Connector port (default 8009) and the Server shutdown port (default 8005) are assigned to free ports. Failure to do so may result in the wrong instance of Tomcat being shutdown, etc. Once you have configured and started Tomcat, you should check for bind errors in

$CATALINA_HOME/logs/catalina.out

Unfortunately, having two servers with the same shutdown port does not always seem to cause a bind error.

To configure the shutdown port, you simply choose the number, and edit the port attribute of the Server element.

To configure the AJP Connector, simply uncomment the definition (search server.xml for “AJP”), and change the port number from 8009 if required, so you have something like the following:

```xml
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8099"
    enableLookups="false" redirectPort="8443" protocol="AJP/1.3" />
```

Set the port number, for the service in $CATALINA_HOME/conf/server.xml. By default, this is on port 8080. You will use the ports defined in your acceptor list above. To set this, look for lines reading:

```xml
<!-- Define a non-SSL HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" maxHttpHeaderSize="8192"
```

and change the port attribute accordingly.

### 5.1.2 Tomcat Manager Login

Configure a manager login for Tomcat (to be used when deploying the acceptor). Edit

$CATALINA_HOME/conf/tomcat-users.xml

and add the following lines inside the tomcat-users element:

```xml
<role rolename="manager"/>
<user username="manager" password="manager" roles="manager"/>
```

Any username and password can be set. Note these down—they’re needed for the Acceptor configuration later on.
5.1.3 Restarting Tomcat

You should now be able to start Tomcat using startup.sh, and shut it down again using shutdown.sh. It’s good to try this now, so that you can check the log files for bind errors, etc. before proceeding further.

You can verify it’s up and running using a web browser, and going to the main page for the server, e.g.:

http://nodex-1.hpc.ncsu.edu:9876/

And you can verify the manager login by clicking on the “Tomcat Manager” link under the “Administration” section on the left of the page, and trying the username/password combination out.

5.2 Building mod_jk

In order to connect Apache HTTPD and Apache Tomcat, we need to install the JK connector. Download links are on:


We recommend JK-1.2.21 (or higher), which fixes an exploit that exists in 1.2.19 and 1.2.20. Versions 1.2.21 and 1.2.22 have been tested and are known to work; versions older than 1.2.19 have not been tried. As information about exploits tends to change, you should check the web page above in any case, for the latest information.

If you’ve got a binary, simply put the file in $HTTPD/modules, then either symbolically link (or rename) the file to mod_jk.so, and you can skip to Section 5.3.

Otherwise, you will need to do something like the following, after downloading and unpacking the tarball.

    cd tomcat-connectors-1.2.21-src
    cd native
    ./configure --with-apxs=$HTTPD/bin/apxs
    cd apache-2.0
    make -f Makefile.apxs
    sudo make -f Makefile.apxs install

The module will now be installed at:

$HTTPD/modules/mod_jk.so

5.3 Linking Apache HTTPD and Tomcat with mod_jk

5.3.1 Tomcat Configuration

1. Shutdown Tomcat (run shutdown.sh), if running.
2. Edit $CATALINA_HOME/conf/server.xml. Add this line inside the Server element, after the other Listener elements.

   `<Listener className="org.apache.jk.config.ApacheConfig" modJk="$HTTPD/modules/mod_jk.so"/>

You must substitute the full path of the root of your Apache HTTPD installation for $HTTP, e.g.

   `<Listener className="org.apache.jk.config.ApacheConfig" modJk="/usr/local/apache2/modules/mod_jk.so"/>

3. We are now going to create the JK configuration file, workers.properties, which belongs in $CATALINA_HOME/conf/jk. First, create any necessary subdirectories, and do a cd:

   ```
   mkdir -p $CATALINA_HOME/conf/jk
   cd $CATALINA_HOME/conf/jk
   ```

4. Now edit the new file workers.properties and paste this in it:

   ```
   workers.tomcat_home=$CATALINA_HOME
   workers.java_home=$JAVA_HOME
   
   worker.list=harc
   
   worker.harc.type=ajp13
   worker.harc.host=localhost
   worker.harc.port=8009
   worker.harc.socket_keepalive=1
   ```

   You must substitute the full paths of CATALINA_HOME and JAVA_HOME in the definitions of workers.tomcat_home and workers.java_home, and also change the value of worker.harc.port to point to the Tomcat AJP Port, if you did not use the default of 8009. This file is read by both Tomcat and Apache (it's pointed to by the mod_jk.conf, which is included into httpd.conf), which is why it needs to say where Tomcat and Java are.

5. Run startup.sh to start Tomcat. The file $CATALINA_HOME/conf/auto/mod_jk.conf will be generated by Tomcat, containing something like:

   ```
   ########## Auto generated on Tue Mar 06 10:26:51 EST 2007##########
   
   <IfModule !mod_jk.c>
   LoadModule jk_module "/home/maclaren/apache2/modules/mod_jk.so"
   </IfModule>
   
   JkWorkersFile "/home/maclaren/apache-tomcat-6.0.10/conf/jk/workers.properties"
   JkLogFile "/home/maclaren/apache-tomcat-6.0.10/logs/mod_jk.log"
   JkLogLevel emerg
   ```

   To further verify that everything is ok, you should look in $CATALINA_HOME/logs/catalina.out, and make sure that there are no warnings, e.g.

   ```
   WARNING: Can't find workers.properties at ...
   ```
5.3.2 Apache HTTPD Configuration

1. Stop apache

   $HTTPD/bin/apachectl stop

2. Add these lines to the end of $HTTPD/conf/httpd.conf

   # mod_jk configuration, to be added at the end of your httpd.conf
   Include $CATALINA_HOME/conf/auto/mod_jk.conf

   but substituting your CATALINA_HOME accordingly, to give a full path, e.g.

   #To be added at the end of your httpd.conf
   Include /Users/jonmaclaren/ws/apache-tomcat-5.5.20/conf/auto/mod_jk.conf

3. Edit the Apache httpd-ssl.conf Add the following line to the inside of the VirtualHost element in $HTTPD/conf/extra/httpd-ssl.conf. You can add it to the end of the VirtualHost element, which is probably at the bottom of the file:

   JkMount /currituck/* harc

   Substitute your acceptor name (the part of the URL following the host and port) for “currituck”.

4. Restart Apache:

   $HTTPD/bin/apachectl start

   If there’s a problem with the module, it’ll get reported here. You should check $HTTPD/logs/error_log too. You should get a notice saying:

   [Tue Mar 06 11:02:51 2007] [notice] Apache/2.2.4 (Unix) mod_ssl/2.2.4
   OpenSSL/0.9.7m mod_jk/1.2.21 configured -- resuming normal operations

5.4 Testing the Configuration

To test the configuration, you can use the same commands (or browser configuration) as in Section 4.4. But, instead of accessing the root document of the web server, you should try to access the directory specified for the Acceptor, e.g.:

   https://nodex-1.hpc.ncsu.edu:9877/currituck/

You must remember the trailing slash on this URL. You should get an error message back, but this will be from Tomcat, if the linkage has been successfully performed.

From the command line, the output will contain something like:

<html><head><title>Apache Tomcat/6.0.10 - Error report</title><style>!
   --H1 {font-family:Tahoma,Arial,sans-serif;color:white;background-color:
   #525D76;font-size:22px;} H2 {font-family:Tahoma,Arial,sans-serif;color:
   white;background-color:#525D76;font-size:16px;} H3 {font-family:Tahoma,
   Arial,sans-serif;color:white;background-color:#525D76;font-size:14px;}</html>
HTTP Status 404 - /currituck/

Status report

The requested resource (/currituck/) is not available.

Apache Tomcat/6.0.10
6 Installing the Acceptors

6.1 Initial Tasks

Before we edit the Acceptor’s configuration files, we must still:

1. Install the JDBC Driver;
2. Install Tomcat’s Ant Jarfile; and
3. Install the CA Certificates for the Acceptor.

6.1.1 Installing the JDBC Driver

Before the Acceptor will be able to access the Persistence Database, you will need to install the JDBC driver for your database into the common library directory of Tomcat. The precise instructions for this vary for the different databases, but in general it is possible to find pre-compiled drivers as Jar files. We give details for MySQL and Oracle. In general, see either of:

5.5 – http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html

depending on your Tomcat version.

MySQL  The JDBC Driver for MySQL is called Connector/J, and can be downloaded from the MySQL website at:

   http://dev.mysql.com/downloads/connector/j/

We initially validated the Acceptors using Connector/J 3.1, but now recommend using Version 5.0 (there was no Version 4). The distribution should contain a previously compiled Jar file, e.g.

   mysql-connector-java-5.0.5-bin.jar

Oracle  Downloads for Oracle's JDBC driver can be found at:


Oracle now distribute files with the correct suffix, e.g. ojdbc14.jar. (Older versions used to have the suffix ‘.zip’ and required renaming.)
Installing the Driver

You can now simply copy the jar file as follows, using one of the following commands:

5.5 – cp -p mysql-connector-java-5.0.5-bin.jar $CATALINA_HOME/common/lib
6.0 – cp -p mysql-connector-java-5.0.5-bin.jar $CATALINA_HOME/lib

You will need to restart Tomcat (shutdown.sh, pause, startup.sh) to make this work properly.

6.1.2 Installing Tomcat’s Ant Jarfile

Before compiling, you’ll need to copy the catalina-ant.jar from Tomcat to $ANT_HOME/lib as shown below. The command should look like:

5.5 – cp $CATALINA_HOME/server/lib/catalina-ant.jar $ANT_HOME/lib
6.0 – cp $CATALINA_HOME/lib/catalina-ant.jar $ANT_HOME/lib

6.1.3 Installing Tomcat’s Ant Jarfile

In order for the Acceptor to be able to sign messages, it will need to have a PKCS#12 keystore containing it's credential. As discussed in Section 2.3, this may or may not be the same credential as used for the Apache instance.

If your credential exists as separate PEM certificate and key files, then you can create a PKCS#12 bundle as follows:

```bash
openssl pkcs12 -export -in server.crt -inkey server.key \
-out currituck.p12 -name "Acceptor Currituck" -certfile server-ca.crt
```

6.1.4 Installing CA Certificates for Java

In addition to the security configuration you did for Apache, you will also need to install the CA Certificates that relate to all Acceptor, Resource Manager, and Apache HTTPD certificates into a Java keystore. Either you can install the certificates in the machine-wide location, which is:

```
$JAVA_HOME/lib/security/cacerts
```

Or you can create a separate keystore, and configure the acceptors to point to this. We recommend the second option, which is reflected in the examples in the rest of this section.

For each certificate, you should first double-check the identity:

```bash
keytool -printcert -v -file /Users/jonmaclaren/cct-ca.crt
```
Then add it, using the following command—but replace the name of the keystore with your own location as needs be (you won't need to use sudo either). Keystores have passwords. If you’ve never looked at the system-wide keystore before, its password will be “changeit”, which is good advice. If you choose to use a new keystore, then the keystore is created the first time you “add” a certificate to it; at that time, you will be prompted for a password (but only once, and it is visible on the command line).

```bash
sudo keytool -import -keystore /home/harc/security/cacerts \
   -alias "cct-ca" -file /Users/jonmaclaren/cct-ca.crt
```

### 6.2 Configuring

There are three files that you will need to configure before building and deploying the acceptors. These are:

1. `$HARC/sample/build.xml`
2. `$HARC/sample/web/META-INF/context.xml` - This tells Tomcat how to access the persistence database to create the JNDI DataSource for the acceptor
3. `$HARC/sample/web/WEB-INF/web.xml` – This tells the acceptor code where it and the other acceptors are located, and also the name of the JNDI DataSource that will be used to access the persistence database.

In this section the accceptor name chosen in Section 2.2 is used often, and is referred to by the shorthand, `<name>`.

#### 6.2.1 Editing build.xml

There are slightly different versions of this file for Tomcat 5.5.x and Tomcat 6.0.x. These are in the distribution as:

- 5.5 – `$HARC/sample/build55.xml`
- 6.0 – `$HARC/sample/build60.xml`

Pick the one that matches your Tomcat version, and copy it to `build.xml`, e.g.

```bash
cd $HARC/sample
cp build60.xml build.xml
```

*If you are only building the Acceptor for someone else to deploy, then you should steps skip 2 & 3 below.*

1. Set the definition of the app.name property, changing ACCEPTOR to the `<name>` of the acceptor.
2. Set the definition of the properties manager.username and manager.password to match what was set up in Section 5.1.
3. Set the definition of the property manager.url to point to the Tomcat server you set up in Section 5.1—check the port!
6.2.2 Editing context.xml

You will need to make the following changes. Wherever ACCEPTOR appears in capitals in context.xml, substitute <name> for ACCEPTOR. This means the following places:

1. Change the attribute docBase to be equal to <name>.war in the Context element
2. Change the prefix attribute in the Logger element
3. Change the name attribute in the Resource element to "jdbc/<name>DB"
4. Set the username attribute in the Resource element to be equal to the database access username chosen in Section 3.
5. Set the password attribute in the Resource element to be equal to the database access password chosen in Section 3.

MySQL Substitute the acceptor name for ACCEPTOR in the url attribute in the Resource element—this should match the name of the persistence database, following the recommendations of Section 3, e.g.

   jdbc:mysql://localhost/acceptor_currituck

This may be slightly different if you are using a non-standard port for your MySQL installation, e.g.

   jdbc:mysql://localhost:9999/acceptor_currituck

Databases other than MySQL There is an additional step here for other databases. In the supplied context.xml, the attribute driverClassName in the Resource specifies the MySQL database driver class name. You should change this to the class name for your database. The database URL will also look different.

Oracle Change this to: oracle.jdbc.OracleDriver, and the database URL will look something like this:

   jdbc:oracle:thin:@127.0.0.1:1521:acceptor_currituck

Note that the last part of the URL is the SID, which refers to the Oracle instance containing the database. Here we assume that this has the same name as the database...

Others For other databases, consult your database's documentation. Also, see either of:

5.5 – http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html
6.2.3 Editing web.xml

This file contains the configuration information for the location of all the acceptors. These are specified as parameters inside the servlet element, each in an init-param element; each of these contain one param-name and one param-value elements.

1. You need to point the Acceptor to the credential it will be signing messages with, and also configure the password needed to open the PKCS#12 bundle. This is controlled by the securityKeyStore and securityKeyStorePassword parameters. These should be set to point to the keystore identified above in Section 6.1.3, e.g:

   `<init-param>
       <param-name>securityKeyStore</param-name>
       <param-value>/home/harc/security/currituck.p12</param-value>
   </init-param>
   
   `<init-param>
       <param-name>securityKeyStorePassword</param-name>
       <param-value>pass</param-value>
   </init-param>`

2. To override the default CA keystore location, you need to set the keystore location in the web.xml file for the acceptors, using the init parameter trustedCACertsStore, e.g.

   `<init-param>
       <param-name>trustedCACertsStore</param-name>
       <param-value>/home/harc/security/cacerts</param-value>
   </init-param>`

   This parameter is commented out in the example given in $HARC/sample/web/WEB-INF/web.xml.

3. The list of acceptors is defined as follows. The parameter acceptorNames gives a comma-separated list of acceptor names, then, for each name, there are two further parameters: acceptor.NAME.https which points to the Acceptor’s URL, as decided in Section 2.2; and acceptor.NAME.dn which contains the Distinguished Name of that Acceptor’s credential, written in the following form:

   `/C=US/O=LSU/OU=CCT/CN=EnLIGHTened HARC Acceptor Currituck`

   Finally, the parameter myAcceptorName is set to the name of this Acceptor, and must match one of the names in acceptorNames.

   This list of parameters might look like the excerpt in Figure 2. You will likely find it saves you time to prepare this part of the web.xml file separately, and paste it into each Acceptor’s web.xml file in turn.

4. Finally, you need to replace ACCEPTOR in the JDBCDataSource parameter, so that the name of the data source is exactly matching the name used in the context.xml file, e.g. jdbc/currituckDB. This change needs to also be made to the res-ref-name element, lower down in the file, inside the resource-ref element.

---

2The default type for the CA Keystore is “jks” for Java Keystore, i.e. something created with keytool. If you want to use a PKCS#12 store, you can do this by setting the init parameter trustedCACertsStoreType to "pkcs12".
Figure 2: Example HARC Acceptor Configuration (web.xml excerpt).
5. A last optional step for EnLIGHTened, to push the ActionsSucceeded messages to MCNC’s GRCNotifier.

```xml
<init-param>
    <param-name>commitNotifierClass</param-name>
    <param-value>edu.lsu.cct.cosched.enlightenedUtil.GRCNotifier</param-value>
</init-param>
<init-param>
    <param-name>GRCNotifier.spoolDirectory</param-name>
    <param-value>/home/harc/GRC-uploads/spool/sample</param-value>
</init-param>
```

This would cause the XML messages to be written into files in `/home/harc/GRC-uploads/spool/sample`.

6.3 Build and Deploy

Finally we build the acceptor, and deploy it.

```
cd $HARC/sample
ant acc-compile
ant install
```

If you need to replace/update an already deployed acceptor, you should issue the extra command `ant remove` before the install command.

If you are not allowed to deploy the webapp yourself, you can issue the following sequence of commands instead:

```
cd $HARC/sample
ant acc-compile
ant acc-dist
```

This will construct a self-contained WAR file in `$HARC/sample/dist` which can be sent to the system administrator for installation.

6.4 Verifying the Installation

To test the configuration, you can use the same commands (or browser configuration) as specified in Section 4.4, but using the following URL:

```
https://nodex-1.hpc.ncsu.edu:9877/currituck/getAcceptors
```

This should return an XML document that lists the endpoints of all the Acceptors. It may not be formatted nicely, but if you get some XML back, this indicates that the Acceptor has been successfully deployed, is receiving messages that are being sent to it, and is able to respond.

To make further tests, we will use a Java program located in `$HARC/poster`. You will need a PKCS#12 bundle containing your client credential. Instructions on creating this from separate key and certificate
PEM files can be found in Section 4.4.2. You will also need a file containing some XML; some examples can be found in the same directory as the code, e.g. timetable.xml. To compile the program, type:

```java
cd $HARC/poster
javac securePoster.java
```

Then, try:

```java
java securePoster jon.p12 password \
https://nodex-1.hpc.ncsu.edu:9877/currituck/ping timetable.xml
```

This will send the XML file to the acceptor, which will send it back to you, inside a pingResponse element. If the required certificates have not been installed in the system-wide CA keystore, then you can specify a different location as follows:

```java
java -Djavax.net.ssl.trustStore=./my-ca-store \
securePoster jon.p12 password \
https://nodex-1.hpc.ncsu.edu:9877/currituck/ping timetable.xml
```

You can also get one acceptor to send this to all others using:

```java
java securePoster jon.p12 password \
https://nodex-1.hpc.ncsu.edu:9877/currituck/pingAll timetable.xml
```

This should send back a list of pingResponse elements inside a pingAllResponse element.

Once you’ve installed all your acceptors, this last command should get a response back from every acceptor, when it is send to any acceptor.

You can also log into MySQL as follows:

```bash
mysql -p -u harc acceptor_currituck
```

Then enter your password, and at the prompt, type:

```sql
select * from logger;
```

You should see at least one entry, like:

```sql
mysql> select * from logger;
+---------+---------------------+--------------------------------+
| entry   | logtime             | message                        |
|---------+---------------------+--------------------------------|
| 1       | 2007-03-07 09:20:59 | Initialised acceptor’s servlet |
+---------+---------------------+--------------------------------+
1 row in set (0.00 sec)
```