Setting Up CAS with Ofbiz 5

This wiki explains how to setup and test CAS-Ofbiz5 integration and testing on a Centos 5.2 box called *"elachi"*. In this configuration Ofbiz and the CAS server are both running on the same box.

Supporting Applications

Install some supporting applications if you don't have them already.

Java 1.6

"Clean" way of installing Java: http://trac.arlnet.com/wiki/ArlBasilSetup-Java#Java1.6.6install

OR a quick-and-dirty way...

Download latest Java JDK RPM from http://java.sun.com for your CPU (32bit or 64bit). Then run the downloaded file. This will unzip and then try and rpm install so you will need root privileges.

```
sudo sh jdk-6u11-linux-x64-rpm.bin (it will unpack and rpm install for you)
```

We then need to tell the system to use Java 1.6 (latest) for running Java programs:

```
alternatives --install /usr/bin/java java /usr/java/latest/bin/java 2
alternatives --config java
```

And compiling Java programs:

```
alternatives --install /usr/bin/javac javac /usr/java/latest/bin/javac 2
alternatives --config javac
```

Next we need to set JAVA_HOME so Ant, Tomcat and Ofbiz know where to find Java:

In ~/.bash_profile add:

```
...
JAVA_HOME=<directory Java lives>
...
export ... JAVA_HOME ...
```

Apache Ant

You will need this to build Ofbiz. Should be able to install it via yum.

In ~/.bash_profile add:

```
...
ANT_HOME=<directory Ant lives>
...
export ... ANT_HOME ...
```

Maven (to build CAS Client and Server)

Should be able to install Maven via yum but if you can't it can be manually install by downloading the latest stable release from http://maven.apache.org/download.html.

Then unpack it and place it somewhere sensible (I like /opt):

```
sudo tar -zxvf apache-maven-2.0.9-bin.tar.gz -C /opt/
```

In ~/.bash_profile add:

```
...
PATH=$PATH:/opt/apache-maven-2.0.9/bin
...
export ... PATH ...
```

Setting up Ofbiz 5

Check Out, Patch and Update Ofbiz Code

In your home directory create a directory to put the Ofbiz code in and pull down revision 682577. *(Why this revision? Because that's the version we had when we created the patch.)*

```
mkdir -p ~/projects/arl
cd mkdir -p projects/arl
svn co -r 682577 http://svn.apache.org/repos/asf/ofbiz/trunk ofbiz
```

If our patch (https://issues.apache.org/jira/browse/OFBIZ-1906) has been applied then just run:

```
svn co http://svn.apache.org/repos/asf/ofbiz/trunk ofbiz
```

and skip to the next section :-)

If our patch has not been applied to the trunk download patch from https://issues.apache.org/jira/browse/OFBIZ-1906:

```
wget https://issues.apache.org/jira/secure/attachment/12387539/security-remoteuser_login.diff -O ~/security-remoteuse
```

Then lets apply the patch (which is in our home dir)

```
patch -b -p0 -i ~/security-remoteuser_login.diff -d ~/projects/arl/ofbiz/
...
patching file framework/common/webcommon/WEB-INF/common-controller.xml
patching file framework/webapp/src/org/ofbiz/webapp/control/LoginWorker.java
patching file framework/security/config/security.properties
```

Ok now that we have applied the patch we can update Ofbiz to the latest reversion.

```
cd ~/projects/arl/ofbiz/
svn update
```

Turn on *Remote User Login* in Ofbiz

CAS overloads the Session.getRemoteUser() method to return a remote user if CAS login was successful. We need to tell Ofbiz to look for users using the getRemoteUser() method (this is basically our patch).

Edit `framework/security/config/security.properties` and uncomment `security.login.http.servlet.remoteuserlogin.allow=true`:

```
...
# -- HttpServletRequest getRemoteUser() based ID (for integrations; uncomment to enable)
security.login.http.servlet.remoteuserlogin.allow=true
...
```

Setup Ofbiz Applications to Use CAS Client (Filter)

We want Ofbiz web applications (accounting, e-commerce etc) to first talk to the CAS server before the goto the Ofbiz code. Once we are talking to Ofbiz the getRemoteUser() method should have a username.

To do this we need to install the latest CAS client jars and configure each applications web.xml file.

Build and Integrate CAS Client into Ofbiz

Download the latest client from: http://www.ja-sig.org/products/cas/downloads/index.html (you want the "JA-SIG CAS Java Client").

This will give you no documentation or jar files... you need to build it :-)

Untar the package and then goto the `cas-client-core` directory:

```
cd cas-client-3.1.3/cas-client-core
```

Inside the `cas-client-core` directory we will build the CAS client jar using Maven:

```
mvn clean package
```

This will create a `target` directory which contains the CAS client jar which we need to copy into Ofbiz so we can reference CAS classes in each application's web.xml:

```
cp target/cas-client-core-3.1.3.jar <ofbiz home>/framework/base/lib/
```

We place the jar in `<ofbiz home>/framework/base/lib/` directory because this is viewable by all applications.

If you want to see the Java documentation for the CAS client run the following in `cas-client-core`:

```
mvn javadoc:javadoc
```

You can then point your browser at the file `cas-client-core/target/site/apidocs/index.html` to read the Javadoc.

Configuring Ofbiz Application's web.xml

For every application you want to have CAS protect you need to add the following to the application's `WEB-INF/web.xml` (changing host to what is appropriate for you setup... Please note we will be setting up traffic to go through Apache so standard ports (80 and 443) are used).

Make sure that for each section (<filter>, <filter-mapping>, <listener>) the CAS stuff stuff comes before the Ofbiz stuff.

Filters:

```
...
        <!-- CAS Filter Stuff -->
        <filter>
                <filter-name>CAS Single Sign Out Filter</filter-name>
                <filter-class>org.jasig.cas.client.session.SingleSignOutFilter</filter-class>
        </filter>

        <filter>
                <filter-name>CAS Authentication Filter</filter-name>
                <filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>
                <init-param>
                        <param-name>casServerLoginUrl</param-name>
                        <param-value>https://elachi/authenticate_au/login</param-value>
                </init-param>
                <init-param>
                        <param-name>serverName</param-name>
                        <param-value>elachi</param-value>
                </init-param>
        </filter>

        <filter>
                <filter-name>CAS Validation Filter</filter-name>
                <filter-class>org.jasig.cas.client.validation.Cas10TicketValidationFilter</filter-class>
                <init-param>
                        <param-name>casServerUrlPrefix</param-name>
                        <param-value>https://elachi/authenticate_au</param-value>
                </init-param>
                <init-param>
                        <param-name>serverName</param-name>
                        <param-value>elachi</param-value>
                </init-param>
        </filter>

        <filter>
                <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
                <filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</filter-class>
        </filter>
        <!-- END: CAS Filter stuff -->
...
 Other filter tags
...
```

Filter-Mappings:

```
...
        <!-- CAS Filter Mapping stuff -->
        <filter-mapping>
                <filter-name>CAS Single Sign Out Filter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>

        <filter-mapping>
                <filter-name>CAS Authentication Filter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>

        <filter-mapping>
                <filter-name>CAS Validation Filter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>

        <filter-mapping>
                <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>
                <url-pattern>/*</url-pattern>
        </filter-mapping>
         <!-- END: CAS Filter Mapping stuff -->
...
 Other filter-mapping tags
...
```

Listeners:

```
...
        <!-- CAS Listener stuff -->
        <listener>
                <listener-class>org.jasig.cas.client.session.SingleSignOutHttpSessionListener</listener-class>
        </listener>
        <!-- END: CAS Listener stuff -->
...
 Other listener tags
...
```

Repeat for each Ofbiz web application you want protected

Configure Ofbiz logout URL

In `framework/common/webcommon/WEB-INF/common-controller.xml`, change to:

```
    <request-map uri="logout">
        <security https="true" auth="true"/>
        <event type="java" path="org.ofbiz.webapp.control.LoginWorker" invoke="logout"/>
        <response name="success" type="url" value="https://elachi/authenticate_au/logout"/>
        <response name="error" type="url" value="https://elachi/authenticate_au/logout"/>
    </request-map>
```

Note: https://elachi/authenticate_au/logout is specific for the CAS server.

CAS Server

Install Tomcat

"Clean" way of installing Tomcat: http://trac.arlnet.com/wiki/ArlBasilSetup-Java#Tomcat5install

OR a quick-and-dirty way...

Download latest stable release from http://tomcat.apache.org/download-60.cgi (download Core tar.gz)

```
sudo tar -zxvf apache-tomcat-6.0.18.tar.gz -C /opt/
```

We are not going to talk to Tomcat server directly. Instead we will have Apache talk to Tomcat via AJP. So in `<tomcat home>/conf/server.xml` we can comment out all the other ports (8080 and 8443). We don't want Ofbiz's embedded Tomcat's AJP (running on 8009) to clash with CAS server's Tomcat so change AJP port to 9009:

```
sudo vim /opt/apache-tomcat-6.0.18/conf/server.xml
```

Start up the server:

```
sudo sh /opt/apache-tomcat-6.0.18/bin/startup.sh
```

Building the CAS Server

The CAS server we are using is customised for ARLnet so you need to check it out of CVS (Maven will pull all the standard CAS libraries from the Internet).

First we need to tell our box how to talk to the CVS server and were the CVS root directory is:

```
export CVS_RSH="ssh"
export CVSROOT=:ext:guy@turmeric:/var/lib/cvs
```

Goto a temporary directory and checkout the Arlnet CAS server code:

```
cvs checkout ArlnetCasServer
```

Edit cas.properties to point to the right hosts and ports:

```
sudo vim ArlnetCasServer/src/main/webapp/WEB-INF/cas.properties
```

Edit deployerConfigContext.xml to point to the right database with valid username and password:

```
sudo vim ArlnetCasServer/src/main/webapp/WEB-INF/deployerConfigContext.xml
```

Then we need to build the server with Maven (make sure your JAVA_HOME is defined):

```
cd ArlnetCasServer
mvn clean package
```

Maven will download all stuff it needs, compile the code and then, in the target directory, package it all into a WAR file which we can deploy to our Tomcat server:

```
sudo cp target/authenticate_au.war /opt/apache-tomcat-6.0.18/webapps/
```

Tomcat should pick up the application without a restart. Check the log `/opt/apache-tomcat-6.0.18/catalina.out` to make sure everything went OK.

Configure Apache

Have a look at http://sivel.net/2007/05/apache-tomcat-centos4/ for connecting Tomcat (embedded in Ofbiz) and Apache on Centos 4 (which does not have mod_proxy_ajp).

If one is using Centos 5, which is what we are using in production, you might prefer to use *mod_proxy* and *mod_proxy_ajp*. In Centos 5 when you install apache it also installs *mod_proxy* and *mod_proxy_ajp* so all you need to do is:

```
sudo yum install httpd-2.2.8-1
```

Then start the server:

```
sudo /sbin/service httpd start
```

Make sure when the machine is restarted Apache automatically comes up:

```
sudo /sbin/chkconfig httpd on
```

We then create a configuration file that knows how to redirect traffic to Ofbiz and CAS when a user hits our server with the name (elachi). We

also tell Apache which SSL certificates we want to use for this domain.

We create the file in `/etc/httpd/conf.d` where apache will pick the configuration automatically (don't need to edit http.conf). Because the configurations are loaded in alphabetical order and we want the Proxy and SSL files read first we prefix the file with zzz so it will be read last:

```
sudo vim /etc/httpd/conf.d/elachi.conf
```

Should look something like:

```
<VirtualHost *>
        ServerName elachi

        ProxyTimeout  300
        ProxyStatus   On
        ProxyPass     /accounting/    ajp://localhost:8009/accounting/ keepalive=On
        ProxyPass     /admin/         ajp://localhost:8009/admin/ keepalive=On
        ProxyPass     /arlnet_images/ ajp://localhost:8009/arlnet_images/ keepalive=On
        ProxyPass     /catalog/       ajp://localhost:8009/catalog/ keepalive=On
        ProxyPass     /content/       ajp://localhost:8009/content/ keepalive=On
        ProxyPass     /doc/           ajp://localhost:8009/doc/ keepalive=On
        ProxyPass     /images/        ajp://localhost:8009/images/ keepalive=On
        ProxyPass     /interface/     ajp://localhost:8009/interface/ keepalive=On
        ProxyPass     /ordermgr/      ajp://localhost:8009/ordermgr/ keepalive=On
        ProxyPass     /partymgr/      ajp://localhost:8009/partymgr/ keepalive=On
        ProxyPass     /shop/          ajp://localhost:8009/shop/ keepalive=On
        ProxyPass     /user/          ajp://localhost:8009/user/ keepalive=On
        ProxyPass     /webtools/      ajp://localhost:8009/webtools/ keepalive=On

        ProxyPass     /authenticate_au/ ajp://localhost:9009/authenticate_au/ keepalive=On

        SSLEngine on
        SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
        SSLCACertificateFile  /etc/httpd/ssl/ca.crt
        SSLCertificateFile    /etc/httpd/ssl/server.crt
        SSLCertificateKeyFile /etc/httpd/ssl/server.key.nopass
</VirtualHost>
```

DON'T PANIC at this point the server won't restart because we haven't set up the SSL certificates. Read on.

Setting up SSL Certificate

We must setup some certificate things in order to get Ofbiz's Java to trust CAS's Java and vice versa.

First we want to set up a self signed certificate ie:

- Create a key (CAK) and certificate (CAC) for fake Certificate Authority (CA)
- Create a key (SK) and Certificate Signing Request (SCSR) for our server
- Have our fake CA sign the CSR with their key and then produce a certificate (SC) for our server.

and then we tell Apache and Java (which is running CAS and Ofbiz) to trust us.

Creating the Certificates

The process described here is from http://www.perturb.org/display/entry/754/.

Goto a directory where you can play (I am using ~/tmp)

1. **CAK** - First lets create the Certificate Authority private key (using openssl):

```
openssl genrsa -des3 -out ca.key 4096
```

2. **CAC** - We then create a certificate that is sign with our own key:

```
openssl req -new -x509 -days 3650 -key ca.key -out ca.crt
```

3. **SK** - Now lets generate the private key for your server

```
openssl genrsa -des3 -out server.key 4096
```

4. **SCSR** - Create a CSR (certificate signing request) which we will later sign with our CA key. Please note that your answers to the questions are not very import except **Common Name** this should be exactly match the name of the server you want to use (in my case "elachi" note not www.elachi or elachi.arlnet.com... doesn't matter as long as consistent):

```
openssl req -new -key server.key -out server.csr
```

5. **SC** - Sign the CSR with the CA key we made earlier:

```
openssl x509 -req -days 3650 -in server.csr -CA ca.crt -CAkey ca.key -set_serial 01 -out server.crt
```

6. Optionally remove the password from your key (if you do this protect your key!!!). We want to do this otherwise every time Apache is started the password needs to be entered.. not useful on development machine.

```
openssl rsa -in server.key -out server.key.nopass
```

Set up SSL on Apache

Now we have the CA certificate (which we want Apache to trust), the server's private key (which we use to encrypt SSL traffic) and the server's certificate (which clients can use) we want to put them somewhere Apache can use them so:

```
sudo mkdir /etc/httpd/ssl
sudo cp ca.crt /etc/httpd/ssl/
sudo cp server.crt /etc/httpd/ssl/
sudo cp server.key.nopass /etc/httpd/ssl/
```

Note that in zzz_elachi.conf we have told Apache where to find these files:

```
...
        SSLEngine on
        SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
        SSLCACertificateFile  /etc/httpd/ssl/ca.crt
        SSLCertificateFile    /etc/httpd/ssl/server.crt
        SSLCertificateKeyFile /etc/httpd/ssl/server.key.nopass
...
```

IMPORTANT The file `/etc/httpd/conf.d/ssl.conf` has a virtual domain defined that overlapes our and has another SSL certificate defined... this causes greif to some newer browsers to open the ssl.conf file and comment out all the lines starting from `<VirtualHost _default_:443>` till (and including) `</VirtualHost>`.

At this point you should be able to start the httpd server:

```
sudo /sbin/service httpd restart
```

Then point your browser at your server i.e. https://elachi/ and tell it to trust the certificate.

Tell Java to Trust Our CA

For Ofbiz and CAS (running in Tomcat), which are both running in Java VMs, to trust us we need to tell Java to trust our fake certificate authority. To do this we add our certificate authority's certificate to Java's trusted CA keystore. Make sure you do this for all the versions of Java you are using i.e. if you are using Java 1.5 for Ofbiz and 1.6 for CAS both versions need to trust our CA:

Import our CA cert (password for keystore is "changeit") into "latest" Java version's CA certificate keystore:

```
sudo /usr/java/latest/bin/keytool -import -trustcacerts -file ~/tmp/ca.crt -alias elachica -keystore /usr/java/latest/
```

It is a good idea to restart CAS (Tomcat) and Ofbiz so they load the new keystore entry.

Now it should all work. Point your browser at something like https://elachi/accouting/ and you should be grilled by CAS and then forwarded to Ofbiz.