

Server Deployment

Customizing Views	2
How to Write an AuthenticationHandler	3
Solving SSL Issues	10

Server Deployment

In this section, you'll find mini-guides about various parts of the deployment process for CAS, including installing, clustering, and customizing.

You can download the complete Server Deployment Guide in [PDF format](#).

We're in the process of moving the documentation located on this site into our Wiki. If you're looking for documentation, please look at the [CAS User Manual](#).

Customizing Views

Existing Views

CAS provides two examples of "skins" out of the box. The default skin is a more complex interface, utilizing CSS. The simple skin is the minimum view needed for CAS to work. The default view is located in `{project.home}/webapp/WEB-INF/view/jsp/default/ui/` while the simple views are located in `{project.home}/webapp/WEB-INF/view/jsp/simple/ui/`.

There are four views that you are required to implement:

- `casConfirmView.jsp` - The Confirmation screen a user will see when they have chosen "warn"
- `casGenericSuccess.jsp` - The screen users will see when they successfully authenticate without coming from a service.
- `casLoginView.jsp` - The screen a user sees when they provide their credentials or if there is an error processing their credentials.
- `casLogoutView.jsp` - The screen users see after they have ended a CAS Single Sign On Session.

Customizing the Views

The easiest way to start is to either make a copy of the default ui or the simple ui and place it under the `{project.home}/webapp/WEB-INF/view/jsp` directory such as `{project.home}/webapp/WEB-INF/view/jsp/{institution}`. Next, edit those JSP pages as needed to customize the UI experience.

After editing the pages, you'll need to let CAS know they exist. This is done by making a copy of `{project.home}/webapp/WEB-INF/classes/default_views.properties` and changes the location of the views to reflect your new views. Finally, update `{project.home}/webapp/WEB-INF/cas-servlet.xml`'s `viewResolver` to reflect your properties file instead of `default_views`.

How to Write an Authentication Handler

The Domain of Authentication

Before we discuss actual APIs and code, it may be helpful to review the concepts of CAS authentication.

Credentials

Credentials are what someone presents to CAS as evidence on the basis of which they may be authenticated. A username, password pair is a Credentials, as is a cryptographic certificate, etc.

Principal

A Principal is an entity that is authenticated.

Coding

AuthenticationHandler

The core interface to implement for your authentication-handling plugin is AuthenticationHandler.

```
/**
 * Validate Credentials support for AuthenticationManagerImpl.
 *
 * Determines that Credentials are valid. Password-based credentials may
 * be
 * tested against an external LDAP, Kerberos, JDBC source. Certificates
 * may be
 * checked against a list of CA's and do the usual chain validation.
 * Implementations must be parameterized with their sources of
```

/server/authenticationhandler/index.xml

information.

*

* Callers to this class should first call supports to determine if the

* AuthenticationHandler can authenticate the credentials provided.

*

* @version \$Revision: 1.11 \$ \$Date: 2005/06/17 13:24:38 \$

*/

```
public interface AuthenticationHandler {
```

```
/**
```

```
* Method to determine if the credentials supplied are valid.
```

```
*
```

```
* @param credentials The credentials to validate.
```

```
* @return true if valid, return false otherwise.
```

```
* @throws AuthenticationException An AuthenticationException can contain
```

```
* details about why a particular authentication request failed.
```

```
*/
```

```
boolean authenticate(Credentials credentials)
```

```
throws AuthenticationException;
```

```
/**
```

```
* Method to check if the handler knows how to handle the credentials
```

```
* provided. It may be a simple check of the Credentials class or something
```

```
* more complicated such as scanning the information contained in the
```

```
* Credentials object.
```

```
*
```

```
* @param credentials The credentials to check.
```

```
* @return true if the handler supports the Credentials, false otherwise.
```

```
*/
```

```
boolean supports(Credentials credentials);
```

```
}
```

You can code directly to this interface.

Our trivial example

Suppose that the credentials we're examining consist of usernames and passwords and that we want to authenticate where the password is the integer representation of the length of the username.

```
/**
 * Authenticates where the presented password is the integer length of
 the
 * username.
 */
public class UsernameLengthAuthnHandler
implements AuthenticationHandler {

public boolean authenticate(Credentials credentials) throws
AuthenticationException {

UsernamePasswordCredentials upCredentials =
(UsernamePasswordCredentials) credentials;

String username = upCredentials.getUsername();
String password = upCredentials.getPassword();

String correctPassword =
Integer.toString(username.length());

return correctPassword.equals(password);
}

public boolean supports(Credentials credentials) {
// we support credentials that bear usernames and passwords
return credentials instanceof UsernamePasswordCredentials;
}
}
```

AuthenticationHandlers that take usernames and passwords

You can extend `AbstractUsernamePasswordAuthenticationHandler` if you're dealing with usernames and passwords.

```
/**
 * Abstract class to override supports so that we don't need to duplicate
 the
 * check for UsernamePasswordCredentials.
 *
 * @version $Revision: 1.12 $ $Date: 2005/06/20 18:15:36 $
 */
public abstract class AbstractUsernamePasswordAuthenticationHandler
implements
AuthenticationHandler, InitializingBean {

/**
 * PasswordEncoder to be used by subclasses to encode passwords for
 * comparing against a resource.
 */
private PasswordEncoder passwordEncoder;

/** Instance of logging for subclasses. */
private Log log = LogFactory.getLog(this.getClass());

/**
 * Method automatically handles conversion to UsernamePasswordCredentials
 * and delegates to abstract authenticateUsernamePasswordInternal so
 * subclasses do not need to cast.
 */
public final boolean authenticate(final Credentials credentials)
throws AuthenticationException {

return
authenticateUsernamePasswordInternal((UsernamePasswordCredentials)
credentials);
}

/**
 * Abstract convenience method that assumes the credentials passed in are
 a
 * subclass of UsernamePasswordCredentials.

```

/server/authenticationhandler/index.xml

```
*  
  
* @param credentials the credentials representing the Username and  
Password  
  
* presented to CAS  
  
* @return true if the credentials are authentic, false otherwise.  
  
* @throws AuthenticationException if authenticity cannot be determined.  
*/  
  
protected abstract boolean authenticateUsernamePasswordInternal(  
final UsernamePasswordCredentials credentials)  
throws AuthenticationException;  
  
public final void afterPropertiesSet() throws Exception {  
if (this.passwordEncoder == null) {  
this.passwordEncoder = new PlainTextPasswordEncoder();  
getLog().info(  
"No PasswordEncoder set. Using default: "  
+ this.passwordEncoder.getClass().getName());  
}  
afterPropertiesSetInternal();  
}  
  
/**  
* Method designed to be overwritten subclasses that need to do  
additional  
* properties checking.  
*  
* @throws Exception if there is an error checking the properties.  
*/  
  
protected void afterPropertiesSetInternal() throws Exception {  
// this is designed to be overwritten  
}  
  
/**  
* Method to return the PasswordEncoder to be used to encode passwords.  
*  
* @return the PasswordEncoder associated with this class.
```

/server/authenticationhandler/index.xml

```
*/

public final PasswordEncoder getPasswordEncoder() {
return this.passwordEncoder;
}

/**
 * Method to return the log instance in order for subclasses to have
access
 * to the log object.
 *
 * @return the logging instance for this class.
 */
public final Log getLog() {
return this.log;
}

/**
 * Sets the PasswordEncoder to be used with this class.
 *
 * @param passwordEncoder the PasswordEncoder to use when encoding
 * passwords.
 */
public final void setPasswordEncoder(final PasswordEncoder
passwordEncoder) {
this.passwordEncoder = passwordEncoder;
}

/**
 * @return true if the credentials are not null and the credentials class
is
 * assignable from UsernamePasswordCredentials.
 */
public final boolean supports(final Credentials credentials) {
return credentials != null
&& UsernamePasswordCredentials.class.isAssignableFrom(credentials
.getClass());
}
}
```

`}`

Our trivial example revisited

Suppose again that the password for any given username is the number of letters in that username.

By extending the abstract class, we can implement this handler more simply. The abstract class handles casting the Credentials to a UsernamePasswordCredentials and handles implementing the supports() method.

```
package org.jasig.cas.authentication.handler.support;

import org.jasig.cas.authentication.handler.AuthenticationException;

import org.jasig.cas.authentication.principal.UsernamePasswordCredentials;

/**
 * Authenticates where the presented password is the integer length of
 * the
 * username.
 */
public class UsernameLengthAuthnHandler
    extends AbstractUsernamePasswordAuthenticationHandler {

    protected boolean authenticateUsernamePasswordInternal(
        UsernamePasswordCredentials credentials)
        throws AuthenticationException {

        String username = credentials.getUsername();
        String password = credentials.getPassword();

        String correctPassword =
            Integer.toString(username.length());

        return correctPassword.equals(password);
    }
}
```

Solving SSL Issues

CAS Server requires SSL. Newcomers to deploying secure web applications (and even old hands) sometimes have difficulty configuring the SSL certificates. This page is intended to provide links and help with troubleshooting SSL in the context of deploying an instance of the Central Authentication Service server.

Generating a certificate

I am guessing that you have a certificate that is either not properly generated or is not installed in your jks keystore.

I am assuming you are using 1.4.x or later, otherwise you'd have to install JSSE separately.

To generate the certificate you could do this (substitute argument values as it suits your system). Further documentation about the JDK keytool is available [on Sun's website](#).

```
%JAVA_HOME%\bin\keytool -delete -alias tomcat -keypass changeit
%JAVA_HOME%\bin\keytool -genkey -alias tomcat -keypass changeit -keyalg
RSA %JAVA_HOME%\bin\keytool -export -alias tomcat -keypass changeit
-file %FILE_NAME% %JAVA_HOME%\bin\keytool -import -file server.crt
-keypass changeit -keystore %JAVA_HOME%\jre/lib/security/cacerts
%JAVA_HOME%\bin\keytool -import -file server.crt -keypass changeit
```

Expiration Date of Certificate

If you want the certificate to be valid for longer than the default amount of time, you can provide an option parameter in the following format:

```
-validity numberOfDays
```

which allows you specify the number of days a certificate is valid for. So in the above example you would use the following command to create the certificate and have it valid for 365 days:

```
%JAVA_HOME%\bin\keytool -genkey -alias tomcat -keypass changeit -keyalg
RSA -validity 365
```

World-readability

K.C. Baltz reminded the List that the public cert files must be world-readable, as noted in the README.

How do I configure Tomcat to use SSL?

Try [there](#) instructions.

Where can I learn more about the keytool?

At the [keyool](#) page.

Certificates for IP addresses will not work

Quite simply, issuing and using certificates that authenticate IP addresses rather than host names will not work. Don't do it. For anything larger than a trivial installation, your CAS server and its clients need real hostnames and certificates that authenticate those hostnames (for proxy ticket functionality). In particular, if you get an error like this in the CAS server log, it means that the SSL callback was specified in terms of an IP address rather than a hostname.

```
2006-03-13 15:34:21,011 INFO
[org.jasig.cas.CentralAuthenticationServiceImpl] - <Granted service
ticket [ST-99-FbOVOK6wBfqgUEowijewlmqpdKGQdhr9qkj-20] for service
[https://134.106.68.83:8443/CASseMailServlet/servlet/TANServlet] for user
[8427890]> 2006-03-13 15:34:21,054 ERROR [org.jasig.cas.util.UrlUtils] -
<javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target>

javax.net.ssl.SSLHandshakeException:
sun.security.validator.ValidatorException: PKIX path building failed:
sun.security.provider.certpath.SunCertPathBuilderException: unable to
find valid certification path to requested target

at
```

/server/ssl/index.xml

```
com.sun.net.ssl.internal.ssl.Alerts.getSSLException(Alerts.java:150)
```

...

```
Caused by: sun.security.validator.ValidatorException: PKIX path building
failed: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
```

at

```
sun.security.validator.PKIXValidator.doBuild(PKIXValidator.java:221)
```

...

```
Caused by: sun.security.provider.certpath.SunCertPathBuilderException:
unable to find valid certification path to requested target
```

at

```
sun.security.provider.certpath.SunCertPathBuilder.engineBuild(SunCertPathBuilder.java:118)
```

...

```
2006-03-13 15:34:21,057 INFO
```

```
[org.jasig.cas.authentication.AuthenticationManagerImpl] -
```

```
<AuthenticationHandler:
```

```
org.jasig.cas.authentication.handler.support.HttpBasedServiceCredentialsAuthenticati
failed to authenticate the user.>
```

```
2006-03-13 15:34:21,057 ERROR
```

```
[org.jasig.cas.web.ServiceValidateController] - <TicketException
```

```
generating ticket for:
```

```
https://134.106.68.83:8443/CASseMailServlet/CasProxyServlet>
```

```
org.jasig.cas.ticket.TicketCreationException:
```

```
error.authentication.credentials.bad
```

at

```
org.jasig.cas.CentralAuthenticationServiceImpl.delegateTicketGrantingTicket(CentralA
```

...

```
Caused by: error.authentication.credentials.bad
```

/server/ssl/index.xml

at

org.jasig.cas.authentication.handler.BadCredentialsAuthenticationException.<clinit>(I