Table of contents

I) Add support for OAuth in CAS server

The *cas-server-support-oauth* module is made to add support for OAuth in CAS server. It allows two modes :

- CAS server can support OAuth protocol as an OAuth client : in this case, CAS authentication can be delegated to an OAuth provider like Facebook, GitHub, Google, LinkedIn, Twitter, Yahoo... or even an another CAS server using OAuth wrapper
- CAS server can support OAuth protocol as an OAuth server : in this case, CAS uses the OAuth wrapper and acts as an OAuth server, communicating through OAuth protocol version 2.0 with OAuth clients.

II) How to add OAuth client support in CAS server ?

A) Add dependency

First step is to add the dependency to the OAuth cas support module in the CAS server webapp *pom.xml* :

```
<dependency>
    <groupId>org.jasig.cas</groupId>
    <artifactId>cas-server-support-oauth</artifactId>
    <version>${project.version}</version>
</dependency>
```

B) Add the OAuth providers needed

An identity provider is an OAuth server which can authenticate user (like Google, Yahoo...) instead of a CAS server. If you want to delegate the CAS authentication to Twitter for example, you have to add an OAuth provider for Twitter. OAuh providers are defined in the Scribe UP library (except the CAS one) :

All the OAuth providers must be declared in *applicationContext.xml* :

```
<bean id="facebook1" class="org.scribe.up.provider.impl.FacebookProvider">
    <property name="key" value="the_key_for_facebook1" />
    <property name="secret" value="the_secret_for_facebook1" />
    <property name="callbackUrl" value="http://mycasserver/login" />
</bean>

<bean id="twitter1" class="org.scribe.up.provider.impl.TwitterProvider">
    <property name="key" value="the_key_for_twitter1" />
    <property name="secret" value="the_secret_for_twitter1" />
    <property name="callbackUrl" value="http://mycasserver/login" />
</bean>

<bean id="google1" class="org.scribe.up.provider.impl.GoogleProvider">
    <property name="key" value="the_key_for_google1" />
    <property name="secret" value="the_secret_for_google1" />
    <property name="callbackUrl" value="http://mycasserver/login" />
</bean>

<bean id="yahoo1" class="org.scribe.up.provider.impl.YahooProvider">
    <property name="key" value="the_key_for_yahoo1" />
    <property name="secret" value="the_secret_for_yahoo1" />
    <property name="callbackUrl" value="http://mycasserver/login" />
</bean>

<bean id="linkedin1" class="org.scribe.up.impl.LinkedInProvider">
    <property name="key" value="the_key_for_linkedin1" />
    <property name="secret" value="the_secret_for_linkedin1" />
    <property name="callbackUrl" value="http://mycasserver/login" />
</bean>

<bean id="github1" class="org.scribe.up.provider.impl.GitHubProvider">
    <property name="key" value="the_key_for_github1" />
    <property name="secret" value="the_secret_for_github1" />
```

```
    <property name="callbackUrl" value="http://mycasserver/login" />
</bean>

<bean id="caswrapper1"
class="org.jasig.cas.support.oauth.provider.impl.CasWrapperProvider20">
    <property name="key" value="the_key_for_caswrapper1" />
    <property name="secret" value="the_secret_for_caswrapper1" />
    <property name="callbackUrl" value="http://mycasserver/login" />
    <property name="serverUrl"
value="http://mycasserverwithoauthwrapper/oauth2.0" />
</bean>
```

For each OAuth provider, the CAS server is considered an OAuth client and therefore should be declared to the OAuth provider. After declaration, a *key* and a *secret* is given by the OAuth provider which has to be defined in the beans (*the_key_for_xxx* and *the_secret_for_xxx* values for *key* and *secret* properties). The *callbackUrl* property must be set to the login url of the CAS server.
For the CAS OAuth wrapping, the *serverUrl* property must be set to the OAuth wrapping url of the other CAS server which is using OAuth wrapping.

C) Add the OAuth action in webflow

In the *login-webflow.xml* file, the *OAuthAction* must be added at the beginning of the webflow. Its role is to intercept callback calls from OAuth providers (like Facebook, Twitter...) after OAuth authentication :

```
      <action-state id="oauthAction">
          <evaluate expression="oauthAction" />
          <transition on="success" to="sendTicketGrantingTicket" />
          <transition on="error" to="ticketGrantingTicketExistsCheck" />
      </action-state>
```

This *OAuthAction* has to be defined in *cas-servlet.xml* with all the identity providers needed :

```
<bean id="oauthAction" class="org.jasig.cas.support.oauth.web.flow.OAuthAction">
    <property name="centralAuthenticationService"
ref="centralAuthenticationService" />
    <property name="providers">
        <list>
            <ref bean="facebook1" />
            <ref bean="twitter1" />
            <ref bean="google1" />
            <ref bean="yahoo1" />
            <ref bean="linkedin1" />
            <ref bean="github1" />
            <ref bean="caswrapper1" />
        </list>
    </property>
</bean>
```

This *OAuthAction* uses the *centralAuthenticationService* bean to play the CAS authentication and references all the identity providers.

D) Add the principal resolver and the handler for OAuth authentication

To be able to authenticate user in CAS server after OAuth authentication by the OAuth provider,

you have to add an *OAuthAuthenticationHandler* in the list of *authenticationHandlers* in *deployerConfigContext.xml* :

```
<property name="authenticationHandlers">
  <list>
    <bean
class="org.jasig.cas.support.oauth.authentication.handler.support.OAuthAuthenti
cationHandler">
      <property name="providers">
        <list>
          <ref bean="facebook1" />
          <ref bean="twitter1" />
          <ref bean="google1" />
          <ref bean="yahoo1" />
          <ref bean="linkedin1" />
          <ref bean="github1" />
          <ref bean="caswrapper1" />
        </list>
      </property>
    </bean>
  </list>
</property>
```

And also add an *OAuthCredentialsToPrincipalResolver* in the list of *credentialsToPrincipalResolvers* in the *deployerConfigContext.xml* :

```
<property name="credentialsToPrincipalResolvers">
  <list>
    <bean
class="org.jasig.cas.support.oauth.authentication.principal.OAuthCredentialsToP
rincipalResolver" />
  </list>
</property>
```

E) Add the metadata populator for OAuth authentication

To be able to retrieve the attributes of the user profile, an *OAuthAuthenticationMetaDataPopulator* needs to be defined in the list of *authenticationMetaDataPopulators* in *deployerConfigContext.xml* :

```
      <bean                                              id="authenticationManager"
class="org.jasig.cas.authentication.AuthenticationManagerImpl">
        <property name="authenticationMetaDataPopulators">
            <list>
                <bean
class="org.jasig.cas.support.oauth.authentication.OAuthAuthenticationMetaDataPo
pulator" />
            </list>
        </property>
```

F) Add links on the login page to authenticate on identity providers

To start authentication on an OAuth provider, links must be added on the login page *casLoginView.jsp* (*ProviderTypeUrl* are automatically created by the *OAuthAction*) :

```
<a href="${FacebookProviderUrl}">Authenticate with Facebook</a> <br />
<br />
<a href="${TwitterProviderUrl}">Authenticate with Twitter</a><br />
<br />
<a href="${GoogleProviderUrl}">Authenticate with Google</a><br />
```

```
<br />
<a href="${YahooProviderUrl}">Authenticate with Yahoo</a><br />
<br />
<a href="${LinkedInProviderUrl}">Authenticate with LinkedIn</a><br />
<br />
<a href="${GitHubProviderUrl}">Authenticate with GitHub</a><br />
<br />
<a href="${CasWrapperProvider20Url}">Authenticate with another CAS server using
OAuth v2.0 protocol wrapper</a><br />
<br />
```

III) Technical presentation of the OAuth client mode

A) Dependencies

1) org.scribe / scribe-up / 1.0.0-SNAPSHOT

This module is based on an open source library called Scribe UP for « Scribe User Profile » : it's a web-oriented extension to Scribe to get user profile after OAuth authentication process.
Source code is here : https://github.com/leleuj/scribe-up. It's available under Apache 2 licence. Current version : 1.0.0-SNAPSHOT is available in Sonatype snapshots repository : https://oss.sonatype.org/content/repositories/snapshots.
The Scribe UP library is based on Scribe (for OAuth communication) and Jackson (for JSON parsing).

2) org.jasig.cas / cas-server-core / ${project.version}

B) Classes

1) OAuthAuthenticationMetaDataPopulator (org.jasig.cas.support.oauth.authentication)

This class is a meta data populator for OAuth authentication. As attributes are stored in *OAuthCredentials,* they are added to the returned principal.

2) OAuthAuthenticationHandler (org.jasig.cas.support.oauth.authentication.handler.support)

This handler authenticates OAuth credential : it uses them to get an access token to get the user profile returned by the provider for an authenticated user.

3) OAuthCredentials (org.jasig.cas.support.oauth.authentication.principal)

This class represents an OAuth credential and (after authentication) a user identifier and attributes.

4) OAuthCredentialsToPrincipalResolver (org.jasig.cas.support.oauth.authentication.principal)

This class resolves the principal id regarding the OAuth credentials : principal id is the type of the provider # the user identifier.

5) CasWrapperApi20 (org.jasig.cas.support.oauth.provider.impl)

This class represents the OAuth API implementation for the CAS OAuth wrapper.

6) CasWrapperProvider20 (org.jasig.cas.support.oauth.provider.impl)

This class is the OAuth provider to authenticate user in CAS wrapping OAuth protocol.

7) OAuthAction (org.jasig.cas.support.oauth.web.flow)

This class represents an action in the webflow to retrieve OAuth information on the callback url which is the webflow url (/login). The oauth_provider and the other OAuth parameters are expected after OAuth authentication. Providers are defined by configuration. The service is stored and retrieved from web session after OAuth authentication.

IV) How to add OAuth server support in CAS server ?

A) Add dependency

First step is to add the dependency to the OAuth cas support module in the CAS server webapp *pom.xml* :

```
<dependency>
    <groupId>org.jasig.cas</groupId>
    <artifactId>cas-server-support-oauth</artifactId>
    <version>${project.version}</version>
</dependency>
```

B) Add the OAuth20WrapperController

To add the *OAuth20WrapperController*, you need to add the mapping between the */oauth2.0/** url and the CAS servlet in the *web.xml* :

```
<servlet-mapping>
    <servlet-name>cas</servlet-name>
    <url-pattern>/oauth2.0/*</url-pattern>
</servlet-mapping>
```

You have to create the controller itself in *cas-servlet.xml* :

```
<bean
  id="oauth20WrapperController"
  class="org.jasig.cas.support.oauth.web.OAuth20WrapperController"
  p:loginUrl="http://mycasserverwithoauthwrapper/login"
  p:servicesManager-ref="servicesManager"
  p:ticketRegistry-ref="ticketRegistry"
  p:timeout="7200" />
```

The *loginUrl* is the login url of the CAS server. The *timeout* is the lifetime of a CAS granting ticket (in seconds, not in millisecondes !)

with its mapping in the *handlerMappingC* bean (*cas-servlet.xml*) :

```
<bean id="handlerMappingC"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/logout">logoutController</prop>

  ........................

      <prop key="/403.html">passThroughController</prop>
      <prop key="/oauth2.0/*">oauth20WrapperController</prop>
    </props>
  </property>
  <property name="alwaysUseFullPath" value="true" />
</bean>
```

C) Add the needed CAS services

One service is need to make the OAuth wrapper works in CAS. It defines the callback url after CAS authentication to return to the OAuth wrapper as a CAS service.
A second service is necessary to register an OAuth client : the *name* and the *description* of the CAS service are the *key* and *secret* of the OAuth client. For each OAuth client, a CAS service needs to be added in configuration.

For the in memory service registry, you add the two services in the *deployerConfigContext.xml* :

```
<bean id="serviceRegistryDao"
class="org.jasig.cas.services.InMemoryServiceRegistryDaoImpl">
  <property name="registeredServices">
    <list>
      <bean class="org.jasig.cas.services.RegisteredServiceImpl">
        <property name="id" value="0" />
        <property name="name" value="HTTP" />
        <property name="description" value="oauth wrapper callback url" />
        <property name="serviceId"
value="http://mycasserverwithoauthwrapper/oauth2.0/callbackAuthorize" />
      </bean>

      <bean class="org.jasig.cas.services.RegisteredServiceImpl">
        <property name="id" value="1" />
        <property name="name" value="the_key_for_caswrapper1" />
        <property name="description" value="the_secret_for_caswrapper1" />
        <property name="serviceId" value="http://mycasserver/login" />
      </bean>
```

If you have one CAS server configured with the *CasWrapperProvider20* (the client) to communicate with a CAS server wrapping OAuth 2.0 protocol (the server), you have the *name* and *description* of the service in CAS « server » matching the *key* and *secret* of the identity provider defined in the CAS « client » :

```
<bean class="org.jasig.cas.services.RegisteredServiceImpl">
  <property name="id" value="1" />
  <property name="name" value="the_key_for_caswrapper1" />
  <property name="description" value="the_secret_for_caswrapper1" />
  <property name="serviceId" value="http://mycasserver/login" />
</bean>

<bean id="caswrapper1"
class="org.jasig.cas.support.oauth.provider.impl.CasWrapperProvider20">
    <property name="key" value="the_key_for_caswrapper1" />
    <property name="secret" value="the_secret_for_caswrapper1" />
    <property name="callbackUrl" value="http://mycasserver/login" />
    <property name="serverUrl"
value="http://mycasserverwithoauthwrapper/oauth2.0" />
</bean>
```

V) Technical presentation of the OAuth server mode

A) General

To reply to OAuth calls, the CAS server has a specific controller (*OAuth20WrapperController*),

listening on a specific url (for example : /oauth2.0/*).
This wrapper delegates the authentication to the standard CAS authentication process (the callback is done by using a specific CAS service which is the callback url). It means that after being authenticated by the OAuth wrapped CAS server, the user is also authenticated in CAS.
The OAuth codes generated on the « authorize » OAuth calls are in fact CAS service tickets and the OAuth access tokens generated on the « access token » OAuth calls are in fact granting tickets.
OAuth client configurations are defined with CAS services : the *key* and *secret* of the OAuth clients have to be defined by the *name* and *description* of a CAS service to make OAuth client be « authorized » to use OAuth CAS server.

B) Classes & interfaces

1) OAuthConstants (org.jasig.cas.support.oauth)

This class has the constants for the OAuth implementation.

2) OAuthUtils (org.jasig.cas.support.oauth)

This class has some usefull methods to output data in plain text, handle redirects or add parameter in url.

3) BaseOAuthWrapperController (org.jasig.cas.support.oauth.web)

This controller is the base controller for wrapping OAuth protocol in CAS. It finds the right sub controller to call according to the url.

4) OAuth20WrapperController (org.jasig.cas.support.oauth.web)

This controller is the main entry point for OAuth version 2.0 wrapping in CAS, should be mapped to something like /oauth2.0/*. Dispatch request to specific controllers : authorize, accessToken...

5) OAuth20AuthorizeController (org.jasig.cas.support.oauth.web)

This controller is in charge of responding to the authorize call in OAuth protocol. It stores the callback url and redirects user to the login page with the callback service.

6) OAuth20CallbackAuthorizeController (org.jasig.cas.support.oauth.web)

This controller is called after successful authentication and redirects user to the callback url of the OAuth application. A code is added which is the service ticket retrieved from previous authentication.

7) OAuth20AccessTokenController (org.jasig.cas.support.oauth.web)

This controller returns an access token which is the CAS granting ticket according to the service and code (service ticket) given.

8) OAuth20ProfileController (org.jasig.cas.support.oauth.web)

This controller returns a profile for the authenticated user (identifier + attributes), found with the access token (CAS granting ticket).