MUSE

MuseKnowledge™ Proxy and OAuth Authentication

MuseGlobal, Inc.
One Embarcadero
Suite 500
San Francisco, CA 94111
415 896-6873
www.museglobal.com

MuseGlobal S.A
Calea Bucuresti
Bl. 27B, Sc. 1, Ap. 10
Craiova, România
40 251-413496
www.museglobal.ro

EduLib, S.R.L.
Calea Bucuresti
Bl. 27B, Sc. 1, Ap. 2
Craiova, România
40 351-420970
www.edulib.com

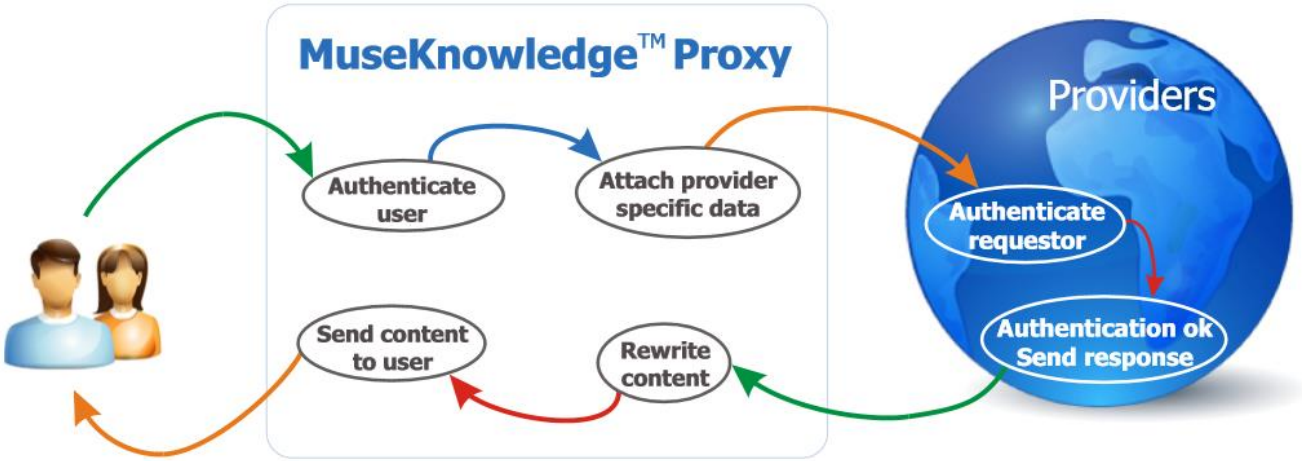
Version: 1.0
Date: 23rd August 2016
Author: EduLib, S.R.L.

MuseKnowledge™ Proxy

Highly customizable multi-platform proxy server

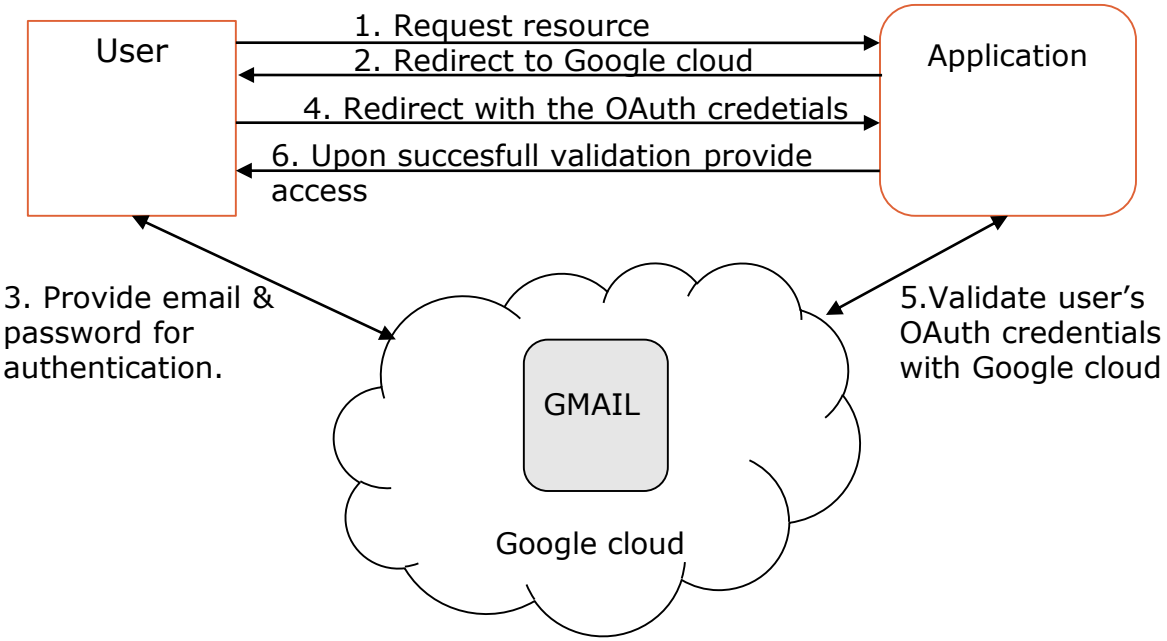
- Easy to use and configure via the MuseKnowledge™ Proxy Administrator Console;
- Gateway to authenticated restricted content;
- Rewriting web server;
- Web Access Management (WAM);
- Proxy server and reverse proxy;
- Supports SAML 2.0 as a Service Provider (SP).

More than 10 years within the Muse Federated Search Platform to manage the authentication to resources and the navigation to full text



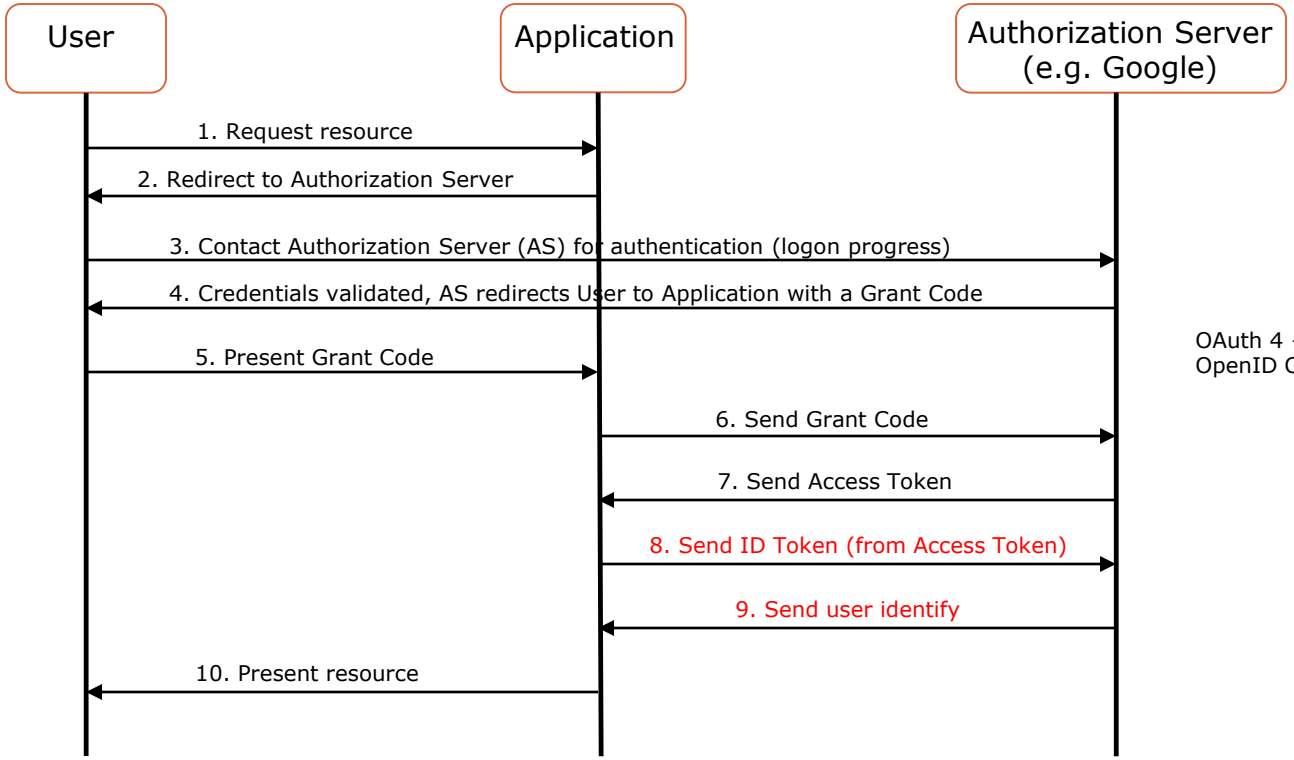
What is OAuth?

- Open standard for authorization;
- Used as a way for Internet users to log in to third party websites using their Google, Facebook, Microsoft, Twitter, etc. accounts without exposing their password; Provides to clients a "secure delegated access" to server resources on behalf of a resource owner; [Wikipedia]



OpenID Connect

- Identity layer built on top of the OAuth 2.0 protocol;
- Implements authentication as an extension to the OAuth 2.0 authorization process, providing information about the end user in the form of an *id_token* (JSON Web Token Profile) that verifies the identity of the user as well as provides basic profile information about the user;



Features and Requirements

Features:

- A wide range of OAuth, OAuth2, OpenID Connect SSO based authentication are supported; OAuth: BitBucket, DropBox, Facebook, Foursquare, Github, Google, LinkedIn, Odnoklassniki, ORCiD, Paypal, Strava, Twitter, Vk, Windows Live, Word Press, Yahoo; OpenID Connect: Google, Azure Active Directory;
- Multiple Service Provider for multiple customer applications (multi-tenancy);
- The OAuth authentication works combined with proxified widgets (forms), *?url=* like links; Other authentications, especially IP authentication (as a sufficient authentication) can be combined with OAuth authentication;
- Walkthrough configuration guide via the MuseKnowledge™ Proxy Administrator Console;
- Possibility of defining custom authentication workflow based on the personal profile items;

Requirements:

- Make sure all server names are configured into *SERVER_NAMES*, including the load balanced name if applicable;
- MuseKnowledge™ Proxy Administrator necessary skills: OAuth and OpenID Connect standards, XML, SSL certificates.

OAuth Configuration Check List

- Activate the internal Servlet Engine where the SSO client component is running (one time action);
- Identify the OAuth/OpenID Connect providers to use; create developer account and applications there to obtain client ID and secret;
- Configure the SSO layer with the new SSO client considering the MuseKnowledge™ Proxy Application ID as well, to ensure multi-tenancy. If no specific client exists and the generic GenericOAuth2StateClient/GenericOAuth2Client does not work then contact MuseKnowledge™ Proxy Support (a new specific client might need to be developed);
- Do tests in MuseKnowledge™ Proxy Administrator Console screen with the configured client to see the JSON raw profile, and pass through the above steps again for other configurations (such as authorizing redirect URL in the provider's developer console);
- MuseKnowledge™ Proxy application wiring and authentication decisions (e.g. sources group based on JSON profile keys).

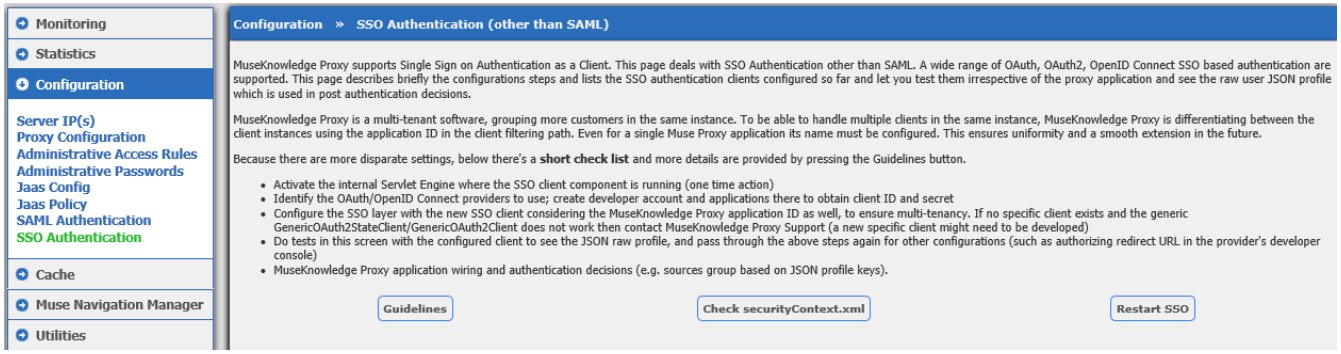
Activate the Internal Servlet Engine

- Edit the file:

  ${MUSE_HOME}/proxy/MuseProxy.xml

  and set to *true* the value in the *SERVLET_ENGINE_ENABLED* node;
- Restart the MuseKnowledge™ Proxy service;
- Jetty Servlet Engine configuration file:

  ${MUSE_HOME}/proxy/ServletEngine.xml
- Accessible only on localhost;
- Restart feature from the MuseKnowledge™ Proxy Administrator Console, *Configuration –> SSO Authentication* left menu, *Restart SSO* button:



Prerequisite Items

- Decide which OAuth or OpenID Connect provider to use;
- Depending on which provider will be used, a developer account and a new "application/project" will need to be created in the provider's administration side;
- After creating a new application/project/client there are two important details that needs to be kept: the client id (or the key) and the client secret - these will be configured in MuseKnowledge™ Proxy;
- Some providers require introduction of the authorized redirect URL, which mainly consist of the MuseKnowledge ™ Proxy server location and the *client_name* parameter, for example:

  http://proxy.domain.com/ssoRWP2/callback?client_name=Google2ClientMuseProxyDevFoundation

- Decide which MuseKnowledge™ Proxy application will be configured for SSO authentication as its ID will be used in configuring the SSO layer (in the file *${MUSE_HOME}/proxy/webcontexts/ssoRWP2/WEB-INF/classes/securityContext.xml*).

Update Configuration Files

- Assumptions: The provider is Google and the MuseKnowledge™ Proxy Application ID is MuseProxyFoundation;
- Edit the file:

  ${MUSE_HOME}/ proxy/webcontexts/ssoRWP2/WEB-INF/classes/securityContext.xml

  - Define a bean with *id="googleSecurityFilterMuseProxyFoundation"*, a reference to *Google2ClientMuseProxyFoundation*, a *security:http* with the pattern */provider/google/MuseProxyFoundation/*** and a reference to *googleSecurityFilterMuseProxyFoundation*;

```
<bean id="googleSecurityFilterMuseProxyFoundation" class="org.pac4j.springframework.security.web.SecurityFilter">
  <property name="config" ref="config"/>
  <property name="clients" value="Google2ClientMuseProxyFoundation"/>
</bean>
<security:http create-session="always" pattern="/provider/google/MuseProxyFoundation/**" entry-point-ref="pac4jEntryPoint">
  <security:custom-filter position="BASIC_AUTH_FILTER" ref="googleSecurityFilterMuseProxyFoundation"/>
</security:http>
```

  - A bean with *id="googleClientMuseProxyFoundation"* which have a name property with value "*Google2ClientMuseProxyFoundation*" must be created.

```
<bean id="googleClientMuseProxyFoundation" class="org.pac4j.oauth.client.Google2Client">
  <property name="name" value="Google2ClientMuseProxyFoundation"/>
  <property name="key" value="{the client id here}"/>
  <property name="secret" value="{the secret here}"/>
  <property name="callbackUrl" value="http://proxy.domain.com/ssoRWP2/callback"/>
</bean>
```

Update Configuration Files

- Add the new client inside *<bean id="clients"* under its *<list>* sub-element;

```
<bean id="clients" class="org.pac4j.core.client.Clients">
  <property name="callbackUrl" value="http://localhost:8080/callback"/>
  <property name="clients">
    <list>
      ...
      <ref bean="googleClientMuseProxyFoundation">
      ...
```

- Make sure that the securityContext.xml file remains XML well formed and restart the SSO;
- Use the corresponding "Test and Obtain JSON Profile" link from the SSO Authentication screen or try to fix the errors reported there;
- Configure the MuseKnowledge™ Proxy Application with SSO, edit AuthenticationGroups.xml and add

```
<AUTHENTICATION>
  <IDENTIFIER>11</IDENTIFIER>
  <LEVEL>requisite</LEVEL>
  <CLASS>com.edulib.muse.proxy.authentication.modules.ProxyLoginModuleSSO</CLASS>
  <HANDLER>
    <CLASS>com.edulib.muse.proxy.authentication.modules.ProxyLoginModuleSSODataHandlerXml</CLASS>
    <!--
    <PARAMETERS>
      <CONFIGURATION_FILE> The configuration file path used by the above handler class. </CONFIGURATION_FILE>
    </PARAMETERS>
    -->
    <PARAMETERS>
      <CONFIGURATION_FILE>${WEB_CONTEXT_HOME}/profiles/login/ProxyLoginModuleSSOGoogle.xml</CONFIGURATION_FILE>
    </PARAMETERS>
  </HANDLER>
</AUTHENTICATION>
```

Final Authentication Decisions

- Have a provider specific named profile, such as *ProxyLoginModuleSSOGoogle.xml*;
- Update the value of the *END_POINT* node accordingly:

<END_POINT>/ssoRWP2/provider/${providerName}/${WEB_CONTEXT_ID}</END_POINT>

where only *${providerName}* must be configured with the value from the pattern attribute used above in *securityContext.xml* as attribute of the corresponding *<security:http*; for example for *"/provider/google/MuseProxyFoundation/*** we will have:

<END_POINT>/ssoRWP2/provider/google/${WEB_CONTEXT_ID}</END_POINT>

- *INPUT* XML element:
  - defines input parameters to be used as global JavaScript variables in the *SCRIPT* section by mapping between a variable id and the friendly name from the JSON profile;

Final Authentication Decisions

- *OUTPUT* XML element:
  - defines the output parameters usable in the Freemarker templates in the MuseKnowledge™ Proxy Application interface;
  - contains *PARAM* elements with *id* and *source* attribute;
  - the *id* represents the name of the property to be further used in the MuseKnowledge™ Proxy Application interface;

```
<OUTPUT>
 <PARAM id="email" source="email"/>
 <PARAM id="displayName" source="displayName"/>
 <PARAM id="imageUrl" source="imageUrl"/>
</OUTPUT>
```

Final Authentication Decisions

- *SCRIPT* XML element:
  - contains under a CDATA construction the JavaScript code;
  - the variables defined in the INPUT section can be used for comparisons for final decision on the authentication;
  - the following global variables have to be used:
    - *succeeded* variable say if authentication pass or not;
    - *logUserID* variable which caries the value of the user identifier to be logged;
    - *sourcesGroupID* selects the source group to be used. If missing or empty, the default group is implied;
    - *errorMessage* variable that store the error message sent back to login module.

Final Authentication Decisions

```
<![CDATA[
                                var succeeded = false;
                                var sourcesGroupID = null;
                                var errorMessage = null;
                                var logUserID = null;
                                var email;
                                var displayName;
                                var imageUrl;
                                var domainWhiteList = ["inf.ucv.ro"];
                                var emailWhiteList = ["david.dascalescu@gmail.com", "auras779@gmail.com"];
                                //var emailWhiteList = ["auras779@gmail.com"];
                                doLogin();
                                function doLogin() {
                                        profile = JSON.parse(jsonProfile);
                                        for (var i=0; i < profile.emails.length; i++) {
                                              if (profile.emails[i].type === 'account') {
                                                email = profile.emails[i].value;
                                              }
                                         }
                                        if (typeof email == 'undefined') {
                                              succeeded = false;
                                              errorMessage = 'No email of type account.';
                                              return;
                                        }
                                        var found = false;
                                        // In case we deal with Google Apps we should have a domain in the JSON response (jsonProfile).
                                        if (typeof profile.domain != 'undefined' && typeof domainWhiteList != 'undefined') {
                                                for (var i=0; i < domainWhiteList.length; i++) {
                                                        if (domainWhiteList[i].toUpperCase() === profile.domain.toUpperCase()) {
                                                                found = true;
                                                                //sourcesGroupID = "1";
                                                                break;
                                                        }
                                                }
                                        }
                                        if (!found) {
                                                for (var i=0; i < emailWhiteList.length; i++) {
                                                        if (emailWhiteList[i].toUpperCase() === email.toUpperCase()) {
                                                                found = true;
                                                                //sourcesGroupID = "2";
                                                                break;
                                                        }
                                                }
                                        }
                                        if (!found) {
                                              succeeded = false;
                                              errorMessage = 'Your account [' + email + '] does not have permission to access this proxy application.';
                                              return;
                                        }

                                succeeded = true;
                                logUserID = email;
                                email = email;
                                displayName = profile.displayName;
                                if (typeof profile.image != 'undefined') {
                                    imageUrl = profile.image.url;
                                }
                                return;
                        }
]]>
```

MUSE

MuseKnowledge™ Proxy and OAuth Authentication