FedoraCommons™

2DC

FEDORA ENHANCED SECURITY LAYER

AUTHENTICATION MODULE

DOCUMENT CONTROL

Author(s):	**Nishen Naidoo**
Quality Assurance:	
Release Version:	**0.3**
Status:	**Draft**
Revision:	**99**

Version	Date	Purpose	Amended by
0.1-Draft	2009-07-20	Initial Draft	Nishen Naidoo
0.2-Draft	2009-07-23	Incorporated comments by Chi Nguyen	Nishen Naidoo
0.3-Draft	2009-08-14	Corrected configuration snippets	Nishen Naidoo

TABLE OF CONTENTS

1 Introduction 1
2 Installation 1
3 Configuring JAAS 2
3.1 Sample JAAS Configuration Files 2
3.1.1 XMLUsersFile Configuration 3
3.1.2 LDAP Configuration 3
3.1.2.1 DIRECT BIND 3
3.1.2.2 BIND-SEARCH-COMPARE 3
3.1.2.3 BIND-SEARCH-BIND 4
3.1.3 Cascading Multiple Authentication Mechanisms 4
4 Configuring the Web Application 5
4.1 Configuring the UserServlet 5
4.2 Configuring the JAAS Authentication Filter 5
4.2.1 Filter Configuration 6
4.2.2 Filter Mapping 7
4.2.2.1 Removing Existing FEDORA Security Filters 7
4.2.2.2 Protecting All Activities 7
4.2.2.3 Protecting Selected Activities 7

1 INTRODUCTION

The objective of this project was to rewrite Fedora's authentication layer to reduce complexity and allowing integration with more authentication mechanisms.

We chose to use Java Authentication and Authorization Services (JAAS) for the authentication implementation as this provides a standard for providing the authentication. By adopting JAAS, we are able to leverage many additional benefits. These can include:

- Cascading authentication.
- Reusability of pre-existing JAAS authentication modules such as Kerberos, AD, CAS. There would be a high level of difficulty in incorporating these authentication mechanisms with the current Fedora authentication architecture.
- Comprehensive documentation for people interested in writing their own JAAS Login Modules.

It is important to note that this package provides only for 'authentication' and not 'authorization'. For authorization you are still required to use the Fedora internal XACML or the Muradora XACML system.

2 INSTALLATION

1. Download the source package from http://www.muradora.org/software/fedora-jaas/.

2. Unzip the package.

3. Run 'ant dist' (it is assumed that you have 'ant' installed). This should generate 2 files for you in the 'dist' directory. The fedora-jaas.war is a demo web application that you can use for testing login modules. The fedora-jaas.jar file is the library that contains the necessary classes for JAAS authentication.

4. Copy the fedora-jaas.jar into your $CATALINA_HOME/webapps/fedora/WEB-INF/lib directory.

5. Copy the sample jaas.conf file from the config directory into $FEDORA_HOME/server/config.

6. Edit $FEDORA_HOME/server/config/jaas.conf file (covered in detail in the 'Configuring JAAS' section).

7. Edit $CATALINA_HOME/webapps/fedora/WEB-INF/web.xml (covered in detail in the 'Configuring the Web Application' section).

3 CONFIGURING JAAS

The default location for the Fedora JAAS configuration file is:
$FEDORA_HOME/server/config/jaas.conf.

This default location can be overridden by specifying an alternative in the Fedora Web Application's web.xml file.

3.1 SAMPLE JAAS CONFIGURATION FILES

A detailed tutorial on the JAAS configuration file can be found at:
http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/tutorials/LoginConfigFile.html

To provide a basic overview, the structure of the JAAS configuration file is

```
application-name
{
      module-class01 mode
      option=value
      option=value
      ...
      option=value;

      module-class02 mode
      option=value
      option=value
      ...
      option=value;
};
```

The application-name can be any name and is referenced by the application performing the authentication. The configuration file can contain multiple application-names, each with different configurations.

Each application section can contain one or more login modules. Each login module has a flag which specifies how it will behave and can also have an unlimited number of options. Login modules are executed in sequence as listed in the application section. There are four possible flags that can be used which affect the behaviour of the login modules. Each login module configuration is terminated by a semi-colon ';'.

The flags for the LoginModule are as follows:

FLAG	DESCRIPTION
Required	The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.
Requisite	The LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).
Sufficient	The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the LoginModule list). If it fails, authentication

	continues down the LoginModule list.
Optional	The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

These are some examples of configurations that people might use.

3.1.1 XMLUSERSFILE CONFIGURATION

This is a basic configuration using the XmlUsersFile authentication module. It authenticates users against the fedora-users.xml file that ships with Fedora as the default authentication mechanism.

```
fedora-auth
{
	fedora.server.jaas.auth.module.XmlUsersFileModule required;
};
```

3.1.2 LDAP CONFIGURATION

When using LDAP authentication there are typically three basic configurations that cover most LDAP deployments.

3.1.2.1 DIRECT BIND

This configuration provides direct binding to an LDAP server for authentication.

```
fedora-auth
{
	fedora.server.jaas.auth.module.LdapModule required
	host.url="ldap://dev01.muradora.org"
	auth.type="simple"
	bind.mode="bind"
	bind.filter="uid={0},ou=people,dc=muradora,dc=org";
};
```

3.1.2.2 BIND-SEARCH-COMPARE

Some LDAP configurations have a 'binduser' that has access to people objects and their one-way encrypted passwords. This configuration allows the 'binduser' to connect to an LDAP server, search for the user object based on the users entered credentials, extract the users password from the user object and identify the encryption scheme used. Using this encryption scheme, the provided user password is then also encrypted and the results compared. A match results in successful authentication.

```
fedora-auth
{
	fedora.server.jaas.auth.module.LdapModule required
	host.url="ldap://dev01.muradora.org"
	auth.type="simple"
	bind.mode="bind-search-compare"
	bind.user="uid=binduser,ou=people,dc=muradora,dc=org"
	bind.pass="somepassword"
	search.base="ou=people,dc=muradora,dc=org"
	search.filter="(uid={0})"
	attrs.fetch="cn,sn,mail,displayName,carLicense";
};
```

3.1.2.3 BIND-SEARCH-BIND

This configuration is almost identical to the bind-search-compare strategy except that instead of finding and comparing the passwords, once a user object is found a bind is executed using that user and the provided password. This configuration is particularly useful for authenticating against Active Directory where user passwords are available from the initial bind and search.

```
fedora-auth
{
	fedora.server.jaas.auth.module.LdapModule required
	host.url="ldap://dev01.muradora.org"
	auth.type="simple"
	bind.mode="bind-search-bind"
	bind.user="uid=binduser,ou=people,dc=muradora,dc=org"
	bind.pass="somepassword"
	search.base="ou=people,dc=muradora,dc=org"
	search.filter="(uid={0})"
	attrs.fetch="cn,sn,mail,displayName,carLicense";
};
```

3.1.3 CASCADING MULTIPLE AUTHENTICATION MECHANISMS

Occasionally, it might be useful to authenticate from multiple sources. You might want to authenticate off an LDAP directory, but have a couple of extra users with admin privileges in the fedora-users.xml file. This is in fact recommended as you can then still get access to your repository in the event of a network failure to your LDAP directory. You also might want to authenticate users from multiple LDAP directories, or any other combination. Achieving this with JAAS is trivial.

The example below demonstrates authenticating first off an LDAP server using a direct bind. If this fails, control is passed on to the XmlUsersFile module to attempt to authenticate with the users credentials there. Note the flags for these modules are set to ‘sufficient’. This means that only one of these modules needs to successfully authenticate a user for authentication to be successful.

```
fedora-auth
{
	fedora.server.jaas.auth.module.LdapModule sufficient
	host.url="ldap://dev01.muradora.org"
	auth.type="simple"
	bind.mode="bind"
	bind.filter="uid={0},ou=people,dc=muradora,dc=org";

	fedora.server.jaas.auth.module.XmlUsersFileModule sufficient;
};
```

4 CONFIGURING THE WEB APPLICATION

The Web Application is configured through the web.xml file that is located at $CATALINA_HOME/webapps/fedora/WEB-INF/web.xml. This file is used to configure the application servlets and servlet filters.

4.1 CONFIGURING THE USERSERVLET

A servlet is provided that produces an XML representation of the authenticated user, along with their attributes. The XML format produced is similar to that of the fedora-users.xml file and is structured as follows:

```
<user id="userid">
      <attribute name="attributename1">
            <value>value1</value>
            <value>value2</value>
      </attribute>
      <attribute name="attributename2">
            <value>value1</value>
      </attribute>
</user>
```

This servlet should be configured to be accessible via http://server:port/fedora/user and **should always be protected by the JAAS authentication filter.**

To configure this servlet you will need to add the following two sections to your web.xml file:

```
<servlet>
      <servlet-name>UserServlet</servlet-name>
      <servlet-class>fedora.server.jaas.UserServlet</servlet-class>
</servlet>

<servlet-mapping>
      <servlet-name>UserServlet</servlet-name>
      <url-pattern>/user</url-pattern>
</servlet-mapping>
```

The purpose of this servlet is to provide applications the ability to access user attributes and authenticate users without the need to implement security infrastructure on the application end. For example, attributes such as email and display names can be obtained by applications without them having to configure and use an LDAP based data/authentication store. This also means that the authentication is handled at a single point rather than at the Fedora end and the application end. This will allow easier development of front end applications and remove the duplication of security infrastructure. It also means that the authentication mechanisms are client system agnostic.

4.2 CONFIGURING THE JAAS AUTHENTICATION FILTER

The JAAS authentication is done through a servlet filter. As such, you will need to configure the servlet filter and map servlets to it. How you do the mapping depends on which activities you need to protect. For instance, you might need to protect everything or you might need to leave APIA activities unprotected. If you are protecting everything, the configuration is simple as there is just a single mapping to perform (explained below). If you have selected servlets to protect, then these will typically have to be mapped on a servlet by servlet basis.

4.2.1 FILTER CONFIGURATION

```
<filter>
   <filter-name>AuthFilterJAAS</filter-name>
   <filter-class>fedora.server.jaas.AuthFilterJAAS</filter-class>

   <init-param>
      <param-name>userClassNames</param-name>
      <param-value>fedora.server.jaas.auth.UserPrincipal</param-value>
   </init-param>

   <init-param>
      <param-name>roleAttributeNames</param-name>
      <param-value>eduPersonEntitlement</param-value>
   </init-param>
</filter>
```

The filter can take several parameters (two of which are shown above). The table below describes the parameters and their defaults.

NAME	DESCRIPTION	DEFAULT
jaas.config.location	Points to the location of the JAAS configuration file.	$FEDORA_HOME/server/config/jaas.conf
jaas.config.name	Specifies the JAAS application name to use from the JAAS configuration file.	fedora-auth
userClassNames	A comma separated list of class names used to identify the class or classes used for user principals. It is advisable to set this to prevent unexpected results. This is in place to provide compatibility with JAAS modules that were designed for container level authentication and is equivalent to the ‘userClassNames’ from the Tomcat JAAS realm.	First principal returned from a Subject
roleClassNames	A comma separated list of class names used to identify role names for the subject. This is in place to provide compatibility with JAAS modules that were designed for container level authentication and is equivalent to the ‘roleClassNames’ from the Tomcat JAAS realm.	No default
roleAttributeNames	A comma separated list of attribute names that represent roles for a user. If for example, a user has an attribute called ‘department’, and you would like to have the value of that attribute as a fedoraRole, then add ‘department’ to this list.	fedoraRole, role (always included)

4.2.2 FILTER MAPPING

It is important to note the sequence of <filter-mapping> tags in your web.xml file. This sequence dictates in which order the filters are to be applied to an incoming request. For the security filter, it is essential that this be at the top of the list (i.e. it is the first filter-mapping that is defined). If you have an authorisation filter as well then that should come after the authentication filter.

4.2.2.1 REMOVING EXISTING FEDORA SECURITY FILTERS

There are currently five or six filters for authentication for Fedora (depending on your configuration):

1. SetupFilter (fedora.server.security.servletfilters.FilterSetup)
2. XmlUserfileFilter (fedora.server.security.servletfilters.xmluserfile.FilterXmlUserfile)
3. LdapFilterForAttributes (fedora.server.security.servletfilters.ldap.FilterLdap)
4. RestApiAuthnFilter (fedora.server.security.servletfilters.FilterRestApiAuthn)
5. EnforceAuthnFilter (fedora.server.security.servletfilters.FilterEnforceAuthn)
6. FinalizeFilter (fedora.server.security.servletfilters.FilterFinalize)

These filters and their associated mappings need to be removed.

4.2.2.2 PROTECTING ALL ACTIVITIES

Once your filter is configured, you need to configure the mappings. This is essentially telling your web application what sort of things you want to pass through this filter.

If you want to protect everything, then it is reasonably simple. You can use a single mapping with a URL pattern:

```
<filter-mapping>
      <filter-name>AuthFilterJAAS</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

This means that ALL requests to your web application need to pass through the AuthFilterJAAS. This is all that is required.

4.2.2.3 PROTECTING SELECTED ACTIVITIES

If you want to protect selected activities, you need to map each of these servlets individually.

SERVLET NAME	CATEGORY	DESCRIPTION
AxisServlet	API-A, API-M	SOAP servlet.
AccessServlet	API-A Lite	Provides access to '/get' operations.
ControlServlet	Admin	Provides application admin capabilities.
DatastreamResolverServlet	API-A Lite?	Datastream resolver.
DescribeRepositoryServlet	API-A Lite	Retrieves repository information.
FieldSearchServlet	API-A Lite	Search capabilities.
GetNextPIDServlet	API-M Lite	Provides a new PID.
GetObjectHistoryServlet	API-A Lite	Obtains object history.
ListDatastreamsServlet	API-A Lite	Lists datastreams for an object.
ListMethodsServlet	API-A Lite	Lists methods for an object.
MethodParameterResolverServlet	API-A Lite?	Method resolver.

OAIProviderServlet	OAI	Provides OAI interface.
ReportServlet	Admin	Provides reporting information.
RISearchServlet	API-A Lite	Provides access to the resource index.
UploadServlet	Special	Provides the ability to upload files to Fedora.
WSDLServlet	Special	Provides the WSDL definition for API-A/API-M.
RestServlet	API-M Lite	Provides REST methods for API-M.

To protect selected servlets you need to map the AuthFilterJAAS servlet to each of the servlets you want to protect. For example, if you wanted to protect the ReportServlet, you would have the following:

```
<filter-mapping>
      <filter-name>AuthFilterJAAS</filter-name>
      <servlet-name>ReportServlet</servlet-name>
</filter-mapping>
```

If you wanted to protect all activities except API-A Lite ones, you'd have a filter mapping for each of the listed servlets that are not in the APIA-A Lite category.

It is imperative that you protect the UserServlet. The mapping below must be in your web.xml file:

```
<filter-mapping>
      <filter-name>AuthFilterJAAS</filter-name>
      <servlet-name>UserServlet</servlet-name>
</filter-mapping>
```