Geomajas static security plugin

Geomajas Developers and Geosparc

Geomajas static security plugin

by Geomajas Developers and Geosparc

1.18.3

Table of Contents

List of Examples

Chapter 1. Introduction

The staticsecurity plugin allows you to define the policies which apply for certain users. The policies are statically defined in a XML file (hence the name). These policies can be linked to users which can also defined in the XML file or which are obtained through code (for example linking to a user database or LDAP store).

This plugin also includes an example which adds a custom policy and makes that available in the SecurityContext.

Chapter 2. Configuration

The configuration of staticsecurity involves the following elements:

- configure the use of staticsecurity as security service.
- configure the way to access the users.
- configure the policies to use.

1. Dependencies

Make sure you include the correct version of the plugin in your project. This can be done either by including a reference to geomajas-dep or the following excerpt (with the correct version) in the dependencyManagement section of your project:

```
 <dependency>
   <groupId>org.geomajas.project</groupId>
   <artifactId>geomajas-project-server</artifactId>
   <version>1.18.3</version>
   <type>pom</type>
   <scope>import</scope>
</dependency>
```

If you are using geomajas-dep, this includes the latest version of the staticsecurity plugin (at the time of publishing of that version). If you want to overwrite the staticsecurity plugin version, make sure to include this excerpt *before* the geomajas-dep dependency.

You can now include the actual dependency without explicit version:

```
<dependency>
    <groupId>org.geomajas.plugin</groupId>
    <artifactId>geomajas-plugin-staticsecurity</artifactId>
</dependency>
```

In case your users are stored in LDAP, you will need to include a dependency on the LDAP module:

```
<dependency>
    <groupId>org.geomajas.plugin</groupId>
    <artifactId>geomajas-plugin-staticsecurity-ldap</artifactId>
</dependency>
```

2. Configure users

Users can be configured either statically or dynamically.

2.1. Static configuration of users

In the bean of type SecurityServiceInfo you can define the users which exist. This allows only password based authentication.

The userId is also used as login. You can add a couple of extra fields to give information about the user.

The password needs to be specified as a base64 encoded MD5 hash. The string to be hashed is a concatenation of

- "Geomajas is a wonderful framework" (without quotes)
- user id

- actual password

There is an online tool which can be used to calculate this has as http://progs.be/md5.html.

The authorizations property allow you to specify the policies, indicating what is allowed for the user.

Example 2.1. Configure users

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.SecurityServiceIn
    <property name="users">
        <list>
            <bean class="org.geomajas.plugin.staticsecurity.configuration.UserI
                <property name="userId" value="luc"/>
                <property name="password" value="b7NMSP1pZN3Hi6nJGVe9JA"/> <!--
                <property name="userName" value="Luc Van Lierde"/>
                <property name="userOrganization" value="triathlon"/>
                <property name="userDivision" value="all distances"/>
                <property name="userLocale" value="nl_BE"/>
                <property name="authorizations">
                    <list>
                        <!-- ... include authorizations / policies here -->
                    </list>
                </property>
            </bean>
        </list>
    </property>
</bean>
```

2.2. Static configuration of users (role-based)

The static security has limited support for role-based user configuration. Particularly, the GetUsersCommand supports querying of users based on the roles they have. To make use of this functionality, static users should be configured by specifying a set of named roles and assigning these roles to the users. This is done by configuring the roles property (versus configuring the authorizations directly). A named role is a named list of authorizations.

Example 2.2. Configure users by role

```
    <bean name="editor" class="org.geomajas.plugin.staticsecurity.configuration
        <property name="authorizations">
            <!-- ... include authorizations / policies here -->
        </property>
    </bean>

    <bean name="viewerA" class="org.geomajas.plugin.staticsecurity.configuratio
        <property name="authorizations">
            <list>
                <bean
                    class="org.geomajas.plugin.staticsecurity.configuration.Lay
                    <property name="commandsInclude">
                        <list>
                            <value>.*</value>
                        </list>
                    </property>
                    <property name="toolsInclude">
                        <list>
                            <value>.*</value>
                        </list>
```

```
                </property>
                <property name="visibleLayersInclude">
                    <list>
                        <value>.*</value>
                    </list>
                </property>
            </bean>
        </list>
    </property>
</bean>

<bean name="viewerB" class="org.geomajas.plugin.staticsecurity.configuratio

<bean class="org.geomajas.plugin.staticsecurity.configuration.SecurityServi
    <property name="excludeDefault" value="true"/>
    <property name="authenticationServices">
        <list>
            <bean class="org.geomajas.plugin.staticsecurity.security.Custom
        </list>
    </property>
    <property name="users">
        <list>
            <bean class="org.geomajas.plugin.staticsecurity.configuration.U
                <property name="userId" value="luc"/>
                <property name="password" value="b7NMSP1pZN3Hi6nJGVe9JA"/>
                <property name="userName" value="Luc Van Lierde"/>
                <property name="roles">
                    <list>
                        <ref bean="editor"/>
                    </list>
                </property>
            </bean>
            <bean class="org.geomajas.plugin.staticsecurity.configuration.U
                <property name="userId" value="marino"/>
                <property name="password" value="kMSqVf2EMWilIKhZyV3dKA"/>
                <property name="userName" value="Marino Van Hoenacker"/>
                <property name="roles">
                    <list>
                        <ref bean="viewerA"/>
                    </list>
                </property>
            </bean>
            <bean class="org.geomajas.plugin.staticsecurity.configuration.U
                <property name="userId" value="empty"/>
                <property name="password" value="fV4OclpOUTCXIXEOx1C6sQ"/>
                <property name="userName" value="Mr. Nobody"/>
                <property name="roles">
                    <list>
                        <ref bean="viewerA"/>
                        <ref bean="viewerB"/>
                    </list>
                </property>
            </bean>
        </list>
    </property>
</bean>
```

2.3. Dynamic configuration of users

The user information can also be determined dynamically by explicitly configuring authentication services. The statically defined users (see previous section) are always automatically included after the configured authentication services.

Below is a simple example from the tests which shows a configuration of an authentication service.

Example 2.3. Configuring a custom authentication service

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.SecurityServiceIn
    <property name="authenticationServices">
        <list>
            <bean class="org.geomajas.plugin.staticsecurity.general.CustomAuthe
                <property name="userId" value="custom"/>
                <property name="password" value="custom"/>
                <property name="authorizationInfo">
                    <bean class="org.geomajas.plugin.staticsecurity.configurati
                        <property name="commandsInclude">
                            <list>
                                <value>.*</value>
                            </list>
                        </property>
                    </bean>
                </property>
            </bean>
        </list>
    </property>
</bean>
```

2.4. Using LDAP to store user information

Below is an example configuration for obtaining user information from an LDAP store.

You have to configure the host and port for the LDAP server. The template to build the user DN[1] from the user id needs to be given. In the template the pair of braces ("{}") is replaced by the actual login. You can also define the attributes which are used to store the various pieces of user information. The given name and surname are concatenated (separated by a space) when both have a value.

The roles property is used to define the authorizations for the roles which are configured in the LDAP store. The keys are the values of the roles attribute, often DN values. You can also define the defaultRole. This is the set of authorizations which is always assigned to all users who can authenticate using LDAP (the authorizations for the roles are added to that set).

Example 2.4. Configuration to use LDAP to authenticate users

```
    <bean name="security.securityInfo" class="org.geomajas.security.SecurityInf
        <property name="loopAllServices" value="false"/>
        <property name="securityServices">
            <list>
                <bean class="org.geomajas.plugin.staticsecurity.security.Static
                <bean class="org.geomajas.plugin.staticsecurity.security.LoginA
            </list>
        </property>
    </bean>
```

[1] distinguished named

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.SecurityServi
    <property name="authenticationServices">
        <list>
            <bean class="org.geomajas.plugin.staticsecurity.ldap.LdapAuthen
                <property name="serverHost" value="localhost"/>
                <property name="serverPort" value="3636" />
                <property name="userDnTemplate" value="cn={},dc=staticsecur
                <property name="givenNameAttribute" value="givenName" />
                <property name="surNameAttribute" value="sn" />
                <property name="localeAttribute" value="locale" />
                <property name="organizationAttribute" value="o" />
                <property name="divisionAttribute" value="ou" />
                <property name="rolesAttribute" value="memberOf" />
                <property name="defaultRole">
                    <list>
                        <bean class="org.geomajas.plugin.staticsecurity.con
                            <property name="toolsInclude">
                                <list><value>.*</value></list>
                            </property>
                        </bean>
                    </list>
                </property>
                <property name="roles">
                    <map>
                        <entry key="cn=testgroup,dc=roles,dc=geomajas,dc=or
                            <list>
                                <bean class="org.geomajas.plugin.staticsecu
                                    <property name="commandsInclude">
                                        <list><value>.*</value></list>
                                    </property>
                                </bean>
                            </list>
                        </entry>
                    </map>
                </property>
            </bean>
        </list>
    </property>
</bean>
```

2.5. User directories

Authentication services can provide support for querying their user base. This is done by implementing the `UserDirectoryService` interface. An authentication service that behaves as a user directory should implement the following method:

- `List<UserInfo> getUsers(UserFilter userFilter);`

This method accepts a user filter and returns a list of users that comply with the filter. An allow-all filter (`AllUserFilter`), logical filters (`AndUserFilter`, `OrUserFilter`) and a role filter (`RoleUserFilter`) are provided. Other filters can be added by subclassing `AbstractUserFilter` and implementing the actual filtering in the authentication service.

For the static authentication service, the following method must be overridden:

- `public boolean evaluate(UserFilter filter, UserInfo userInfo)`

For the LDAP authentication service, the following method provides a similar hook:

- `public Filter convert(UserFilter filter)`

3. Configure policies

The policies are configured by providing a list of authentications. Several ways exist, but they all start from the base layer policies.

3.1. Base layer policies policies

The base policies which always need to be provided include

- which commands can be executed
- which tools can be included in the UI
- CRUD rights which are available for the layers

For each of these policies, you can use regular expressions to indicate either what to include or what to exclude. By default you have no rights. You have to indicate what is included and can then refine that list by excluding some stuff from what was included.

The base configuration to allow everything is displayed below:

Example 2.5. Base layer policies

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.LayerAuthorizatio
    <property name="commandsInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="toolsInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="visibleLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="updateAuthorizedLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="createAuthorizedLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="deleteAuthorizedLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
</bean>
```

You can explicitly configure which commands can be executed by that user.

Example 2.6. Command policies

```
<property name="commandsInclude">
    <list>
        <value>command.MarinoLoggedIn</value>
    </list>
</property>
```

You can also specify the tools which are allowed to be displayed in the UI.

Example 2.7. Tool policies

```
<property name="toolsInclude">
    <list>
        <value>.*</value>
    </list>
</property>
<property name="toolsExclude">
    <list>
        <value>Zoom.*</value>
    </list>
</property>
```

Importantly, you can configure the CRUD rights for each of the layers. You can specify for each layer, using include and exclude regular expressions, whether the user

- can view the layer
- add features in the layer
- edit features for the layer
- delete features from the layer

Example 2.8. Layer policies

```
<property name="visibleLayersInclude">
    <list>
        <value>roads</value>
        <value>rivers</value>
    </list>
</property>
<property name="createAuthorizedLayersExclude">
    <list>
        <value>.*</value>
    </list>
</property>
<property name="updateAuthorizedLayersExclude">
    <list>
        <value>.*</value>
    </list>
</property>
<property name="deleteAuthorizedLayersExclude">
    <list>
        <value>.*</value>
    </list>
```

```
</property>
```

3.2. Configure policies based on an area or geometry

Using AreaAuthorizationInfo you can make your policies more granular by defining the are in which the CRUD operations are allowed. You first have to define the normal layer rights. You can then make this more specific by setting the geometry for the rights in the layers property, setting the area for each of the CRUD operations in for the specific layer. The actual geometry is specified using WKT [http://en.wikipedia.org/wiki/Well-known_text].

Example 2.9. Area policies

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.AreaAuthorization
    <property name="visibleLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="updateAuthorizedLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="createAuthorizedLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="deleteAuthorizedLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="layers">
        <map>
            <entry key="beans">
                <bean class="org.geomajas.plugin.staticsecurity.configuration.La
                    <property name="visibleArea"
                              value="MULTIPOLYGON(((1 0,1 10,10 10,10 0,1 0)))"
                    <property name="updateAuthorizedArea"
                              value="MULTIPOLYGON(((4 0,4 10,10 10,10 0,4 0)))"
                    <property name="createAuthorizedArea"
                              value="MULTIPOLYGON(((1 0,1 5,10 5,10 0,1 0)))" /
                    <property name="deleteAuthorizedArea"
                              value="MULTIPOLYGON(((1 0,1 2,10 2,10 0,1 0)))" /
                </bean>
            </entry>
        </map>
    </property>
</bean>
```

3.3. Configure policies using a layer filter

Using the LayerFilterAuthorizationInfo authorization bean you can add a ECQL filter to a layer to limit the features which can be read.

This is done using the filters property which contains a map where the key is the (server) layers id and the value is the filter string.

Example 2.10. Filter on layer policy

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.LayerFilterAuthor
    <property name="visibleLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="filters">
        <map>
            <entry key="beans" value="stringAttr='bean2'"/>
        </map>
    </property>
</bean>
```

3.4. Configure policies for individual features

You can also be specific about which features in a layer are accessible by the user. This can be done using the FeatureAuthorizationInfo authorization bean.

Using the layers property, you can determine which CRUD rights are allowed for which features. The property contains a map with the (server) layer id as key and a LayerFeatureAuthorizationInfo bean as value. This allows you to select the features using include and exclude rules on the feature id. As this only allows you to select on feature ids, consider this bean as an example for building more powerful feature selection policies.

Example 2.11. Feature specific policy

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.FeatureAuthorizat
    <property name="visibleLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="layers">
        <map>
            <entry key="beans">
                <bean class="org.geomajas.plugin.staticsecurity.configuration.La
                    <property name="createAuthorized" value="false"/>
                    <property name="visibleIncludes">
                        <list>
                            <value>.*</value>
                        </list>
                    </property>
                    <property name="visibleExcludes">
                        <list>
                            <value>2</value>
                        </list>
                    </property>
                    <property name="updateAuthorizedIncludes">
                        <list>
                            <value>.*</value>
                        </list>
                    </property>
                    <property name="updateAuthorizedExcludes">
                        <list>
                            <value>2</value>
                            <value>3</value>
```

```
                    </list>
                </property>
                <property name="deleteAuthorizedIncludes">
                    <list>
                        <value>1</value>
                    </list>
                </property>
            </bean>
        </entry>
    </map>
</property>
</bean>
```

3.5. Configure policies for individual attributes

The most specific access policies can be built using the AttributeAuthorizationInfo authorization bean. This allows you to specify which attributes can be used for specific features.

Using the layers property, you can determine which CRUD rights are allowed for which features. The property contains a map with the (server) layer id as key and a LayerAttributeAuthorizationInfo bean as value. This allows you to select the attributes using include and exclude rules on the attribute name. You can add a "@featureId" suffix to the regular expression to limit the attribute to a features (using a regular expression on the feature id). As this only allows you to select on feature ids, consider this bean as an example for building more powerful attribute selection policies.

Example 2.12. Attribute specific policy

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.AttributeAuthoriza
    <property name="visibleLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="updateAuthorizedLayersInclude">
        <list>
            <value>.*</value>
        </list>
    </property>
    <property name="layers">
        <map>
            <entry key="beans">
                <bean class="org.geomajas.plugin.staticsecurity.configuration.La
                    <property name="readableIncludes">
                        <list>
                            <value>.*</value>
                        </list>
                    </property>
                    <property name="readableExcludes">
                        <list>
                            <value>booleanAttr</value>
                            <value>stringAttr@2</value>
                        </list>
                    </property>
                    <property name="writableIncludes">
                        <list>
                            <value>.*</value>
                        </list>
                    </property>
```

```
                    <property name="writableExcludes">
                        <list>
                            <value>stringAttr</value>
                            <value>integerAttr@2</value>
                        </list>
                    </property>
                </bean>
            </entry>
        </map>
    </property>
</bean>
```

Chapter 3. How-to

1. Evaluating directly configured users first

Normal behavior is that the users which a directly configured in the SecurityServiceInfo object users field are searched after the services which are configured in the authenticationServices field. This can be very useful for testing, for example when the remote service is not available (and the system waits for times). This can be changed by using a configuration like

Example 3.1. Evaluating configured users first

```
<bean class="org.geomajas.plugin.staticsecurity.configuration.SecurityServiceIn
    <!-- LDAP authentication AFTER configured users -->
    <property name="authenticationServices">
        <list>
            <bean class="org.geomajas.plugin.staticsecurity.security.StaticAuth
            <bean class="org.geomajas.plugin.staticsecurity.ldap.LdapAuthentica
                <!-- ..... LDAP settings -->
            </bean>
        </list>
    </property>
</bean>
```