PERMIS Authorization in GT4

Version	Date	Comments
0.1	17 February 2009	Stijn Lievens initial version
0.2	26 February 2009	David Chadwick updates
0.3	2 April 2009	Stijn Lievens. Updates.
0.4	18 May 2009	Stijn Lievens. Minor updates after Linying's comments.

Contents

1 Abbreviations

Throughout this guide the following abbreviations are used:

- AA = attribute authority
- AC = X.509 attribute certificate
- DN = X.500 distinguished name
- GT4 = Globus Toolkit v 4
- PDP = policy decision point
- PIP = policy information point
- PKC = X.509 public key certificate

2 GT4 Very Quick Start Guide

Goal: this section aims to reproduce the steps necessary to protect the shutdown service so that only users possessing the role (permisRole, RemoteAdministrator) can use GT4's shutdown service.

Assumptions: You have a freshly installed GT4 toolkit and you also have your own PKC (i.e. the user that is going to try and use the shutdown service) correctly installed.

Step 1: Give yourself the (permisRole, RemoteAdministrator) privilege. To do this use the ACM tool, version 5_0_x, and sign the AC with the default signing key (`permis.p12`). Make sure you put the correct distinguished name into the AC; this has to be the subject name that is present in your PKC. Put the resulting AC somewhere so that it is readable for user `globus` (who is going to start the container). We will assume the location:
`/home/globus/shutdown/RemoteAdministrator.ace`

Step 2: Download and unzip the `permisAuthzGT4_5_0_x.zip` file. Unzip this file and as user `globus` run
`$$ globus-deploy-gar permisAuthzGT4.gar.`
This will install the necessary libraries into the toolkit. Remember
`$$ globus-undeploy-gar permisAuthzGT4`
will undo the previous operation.

Step 3: Prepare the PERMIS configuration files and policies. A policy that can be used is the following:

```
$$ cat /home/globus/shutdown/policy.xml

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X.509_PMI_RBAC_Policy>
<X.509_PMI_RBAC_Policy OID="minimalShutdownPolicy">
  <SubjectPolicy>
    <SubjectDomainSpec ID="SubjectDomain">
      <Include LDAPDN=""/>
    </SubjectDomainSpec>
  </SubjectPolicy>
  <RoleHierarchyPolicy>
    <RoleSpec OID="1.2.826.0.1.3344810.1.1.14" Type="permisRole">
      <SupRole Value="RemoteAdministrator"/>
    </RoleSpec>
  </RoleHierarchyPolicy>
  <SOAPolicy>
    <SOASpec ID="TheSOA" LDAPDN=""/>
  </SOAPolicy>
  <RoleAssignmentPolicy>
    <RoleAssignment>
      <SubjectDomain ID="SubjectDomain"/>
      <RoleList>
        <Role Type="permisRole" Value=""/>
      </RoleList>
      <Delegate Depth="0"/>
      <SOA ID="TheSOA"/>
      <Validity/>
    </RoleAssignment>
  </RoleAssignmentPolicy>
  <TargetPolicy>
    <TargetDomainSpec ID="ShutDownService">
      <Include URL="http://localhost/wsrf/services/ShutdownService"/>
      <Include URL="https://localhost/wsrf/services/ShutdownService"/>
    </TargetDomainSpec>
  </TargetPolicy>
  <ActionPolicy>
    <Action ID="shut" Name="shutdown"/>
  </ActionPolicy>
  <TargetAccessPolicy>
    <TargetAccess>
      <RoleList>
        <Role Type="permisRole" Value="RemoteAdministrator"/>
      </RoleList>
      <TargetList>
        <Target>
          <TargetDomain ID="ShutDownService"/>
          <AllowedAction ID="shut"/>
        </Target>
      </TargetList>
```

```
    </TargetAccess>
  </TargetAccessPolicy>
</X.509_PMI_RBAC_Policy>
```

Write a configuration file `/home/globus/shutdown/init.cfg` with the following contents:

```
# this tells where the policy is located
#
ini: xml=/home/globus/shutdown/policy.xml
#
# this tells us which attribute certificate to use. Multiple
# attribute certificates can be specified by repeating this line
...: ac=/home/globus/shutdown/RemoteAdminstrator.ace
# this tells the PERMIS PDP to initialize
...: init
```

This file should also be readable by user `globus`.

Step 4: Change the configuration for the shutdown service. Update the file `$GLOBUS_LOCATION/etc/globus_wsrf_core/server-config.wsdd` so that the part concerning the `ShutdownService` reads:

```
<service name="ShutdownService" provider="java:RPC"
        use="literal" style="document">
        <parameter name="allowedMethods" value="*"/>
        <parameter name="className"
            value="org.globus.wsrf.container.ShutdownService"/>
        <wsdlFile>
           share/schema/core/management/shutdown_service.wsdl
        </wsdlFile>
        <parameter name="securityDescriptor"
          value="etc/permisAuthzGT4/permis-security-descriptor.xml"/>
        <parameter name="permisAuthz-customConfig"
            value="/home/globus/shutdown/init.cfg"/>
</service>
```

Step 5: Test it: in one terminal do (as user `globus`)

```
$$ globus-start-container
```

In another terminal do:

```
$$ grid-proxy-init
$$ globus-stop-container
```

You should now see the effect in the first terminal, whilst the second terminal simply returns from the command globus-stop-container.

If you comment out the line

```
'...: ac=/home/globus/shutdown/RemoteAdminstrator.ace'
```

from your configuration file then, upon trying to stop the container again, you will get an authorization denied message. This is because you don't have the necessary attributes to use the shutdown service.

3 Protecting the Mathservice Using PERMIS Authorization

Here, we describe how to protect the Mathservice using PERMIS authorization.

Although we hope that the current guide is sufficient, having a basic understanding of the workings of Globus Toolkit authorization framework may be beneficial. A good introduction can be found in the article: “Authorization processing for Globus Toolkit Java Web services” by Tim Freeman and Rachana Ananthakrishnan. The article may be found at the following address: http://www.grid.org/content/authorization-processing-globus-toolkit-java-web-services .

3.1 Assumptions

We assume that you have a working installation of Globus Toolkit version 4.0.x. Furthermore, we assume that you have successfully installed the secure math service used in the GT4 programmer's tutorial into your container.

The source code to install the MathService can be downloaded from here: http://gdp.globus.org/gt4-tutorial/

In order to install the secure math service into your system you have to follow the instructions in section 11.4 of the GT4 programmer's tutorial.

Also, make sure that you compile the `Client_GSISecMsg_Signed` class as this is the client that we will be using.

As is customary, the location of the Globus installation is referred to as `$GLOBUS_LOCATION`.

3.2 Basic Configuration

3.2.1 Globus Configuration Files

First, we will change the security descriptor of the math service to use the PERMIS PDP to make authorization decisions. The security descriptor for the math service is located at `$GLOBUS_LOCATION/etc/org_globus_examples_services_security_first/security-config-first.xml`

Change the contents of this file so that it reads:

```
<?xml version="1.0"?>
<securityConfig xmlns="http://www.globus.org">
    <auth-method>
        <GSITransport/>
        <GSISecureConversation/>
        <GSISecureMessage/>
    </auth-method>
```

```
    <authz value="permisAuthz:issrg.gt4.PermisPDP/>
</securityConfig>
```

The main point here is that we will use the `issrg.gt4.PermisPDP` class to make authorization decisions.

Note that the `security-config-first.xml` file is pointed to from within the `server-config.wsdd` file residing in the same directory. So, alternatively, you can create a new file with the above content and point to it from the deployment descriptor file `server-config.wdd`. This is done using the parameter named `securityDescriptor`.

Every Policy Decision Point (PDP) has to be loaded with the rules (or policy) that govern its decision making process and the `PermisPDP` class is no exception. We will store the configuration information in a file and point to this file from inside the deployment descriptor of the service. i.e., you will have to add the following line to the file `server-config.wsdd`:

```
<parameter name="permisAuthz-customConfig"
    value="etc/org_globus_examples_services_security_first/init.cfg"/>
```

Note how the part before the dash in the <`parameter name` (i.e. `permisAuthz`) which is called the scope, matches the part before the colon in the `<authz value` in the security descriptor file. You can choose any identifier you like here, as long as they match up. The part following the dash (i.e. `customConfig`) is fixed and cannot be configured.

The file pointed to by the `customConfig` parameter should contain the necessary information to locate the policy and may also contain additional information about repositories that are going to be used to pull users' credentials from. A complete description of the format of this configuration file is given in the Appendix.

For now, it is probably sufficient to use a configuration similar to the one used in the example for the shutdown policy.

3.2.2 PERMIS Policy

The PERMIS Policy basically describes **who** can perform which **operation** on what **resource** and optionally specifies conditions and obligations.

In this case the resource we are trying to protect is the Mathservice which is identified by its end point reference. This end point reference is most easily found by simply starting the container and looking it up in the list that is output by the container. Starting the container is done by issuing the command

```
$$ $GLOBUS_LOCATION/bin/globus-start-container
```

Secondly, the operations available for a certain service can be found from the WSDL file describing the service. Note also that once you have decided to use PERMIS authorization for even a single operation from a web service, all its available operations have to be mentioned in the PERMIS policy. Any operation not mentioned will result in a 'permission denied'. This is because PERMIS uses a *"deny all, except"* policy.

Let's assume we would like to protect the math service so that only users who possess the role (permisRole, MathServiceUser) can access its operations. For simplicity, we will not be concerned with the question who issued this particular role to the user although this is most certainly an important consideration in real life applications.

The PERMIS policy that achieves this would be very similar to the one used to protect the Shutdown service. You should include a new `<TargetDomainSpec>` element that gives the URLs of the end point of the Math service. You should also include the available actions in the `<ActionPolicy>` element. In this case the available actions are 'add', 'subtract', 'divide', 'multiply' and 'GetResourceProperty'. Note that it is *not* necessary to include the arguments of these actions. In fact, if you were to include the arguments then you have to use an additional Policy Information Point (PIP) to get it to work (see Section 3.3). If the authorization decisions will not be based on the value of the arguments to the actions we recommend that you leave them out. Of course, the actual argument will still be available to the web service, it's only the PERMIS PDP that will blissfully unaware of them.

Finally, you have to add a new `<TargetAccess>` element so that users possessing the (permisRole, MathServiceUser) role can perform the various operations.

The complete policy would look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X.509_PMI_RBAC_Policy>
<X.509_PMI_RBAC_Policy OID="MathServicePolicy">
<SubjectPolicy>
  <SubjectDomainSpec ID="SubjectDomain">
    <Include LDAPDN=""/>
  </SubjectDomainSpec>
</SubjectPolicy>
<RoleHierarchyPolicy>
  <RoleSpec OID="1.2.826.0.1.3344810.1.1.14" Type="permisRole">
    <SupRole Value="MathServiceuser"/>
  </RoleSpec>
</RoleHierarchyPolicy>
<SOAPolicy>
  <SOASpec ID="TheSOA" LDAPDN=""/>
</SOAPolicy>
<RoleAssignmentPolicy>
  <RoleAssignment>
   <SubjectDomain ID="SubjectDomain"/>
     <RoleList>
       <Role Type="permisRole" Value=""/>
     </RoleList>
     <Delegate Depth="0"/>
     <SOA ID="TheSOA"/>
     <Validity/>
  </RoleAssignment>
</RoleAssignmentPolicy>
<TargetPolicy>
  <TargetDomainSpec ID="MathService">
    <Include
URL="http://localhost/wsrf/services/examples/security/first/MathService"/>
    <Include
URL="https://localhost/wsrf/services/examples/security/first/MathService"/>
  </TargetDomainSpec>
</TargetPolicy>
<ActionPolicy>
  <Action ID="add" Name="add"/>
```

```
  <Action ID="divide" Name="divide"/>
  <Action ID="subtract" Name="subtract"/>
  <Action ID="multiply" Name="multiply"/>
  <Action ID="getResourceProperty" Name="getResourceProperty"/>
</ActionPolicy>
<TargetAccessPolicy>
  <TargetAccess>
    <RoleList>
      <Role Type="permisRole" Value="MathServiceUser"/>
    </RoleList>
    <TargetList>
      <Target>
        <TargetDomain ID="MathService"/>
        <AllowedAction ID="add"/>
        <AllowedAction ID="divide"/>
        <AllowedAction ID="subtract"/>
        <AllowedAction ID="multiply"/>
        <AllowedAction ID="getResourceProperty"/>
      </Target>
    </TargetList>
  </TargetAccess>
</TargetAccessPolicy>
</X.509_PMI_RBAC_Policy>
```

3.2.3 Test It

Now you should be able to test this. Create an X.509 AC for which the holder matches the subject of the PKC that was used to generate your proxy certificate. You can find out the subject of your PKC by doing:

```
$$ grid-cert-info -file usercert.pem -subject
```

The X.509 AC is most easily created using the ACM software which is available from http://sec.cs.kent.ac.uk/permis/downloads/Level1/acm.shtml Since in this case the issuer of the X.509 AC is irrelevant (as far as the PERMIS policy is concerned) you might as well sign it using the default key that is supplied with the ACM software, which is contained in the file `permis.p12`. The password to open this particular key store is `l3tM3InNow` . It should be noted that the first character is the lowercase letter L and not the digit 1.

By adding or removing this particular AC from the configuration file (as per the Shutdown service) you should now be able to see the effect of the PERMIS authorization.

3.3 More Advanced Configuration – Using Action Parameters

The above is fine if we don't want to use the action arguments in the decision making progress. But what if we do? PERMIS policies provide a way of doing this by adding an IF statement to a `<TargetAccessPolicy>`.

Assume we would like to protect the add operation so that a MathServiceUser can only invoke it if the add argument is less than 100 (i.e. IF permisRole = MathServiceUser AND add.argument < 100 THEN grant add access to MathService)

First, we have to give the add action an argument. This is done as follows in the PERMIS policy:

```
<Action ID="add" Name="add">
  <Argument Name="add_argument" Type="Integer"/>
```

```
</Action>
```

Next, we have to define a new <TargetAccess> element for the add action. This is done in the following way:

```
<TargetAccess>
 <RoleList>
   <Role Type="permisRole" Value="MathServiceUser"/>
 </RoleList>
 <TargetList>
   <Target>
     <TargetDomain ID="MathService"/>
     <AllowedAction ID="add"/>
   </Target>
 </TargetList>
 <IF>
   <LT>
     <Arg Name="add_argument" Type="Integer"/>
     <Constant Type="Integer" Value="100"/>
  </LT>
 </IF>
</TargetAccess>
```

Of course, the 'add' action has to be removed from the list of allowed actions in the <TargetAccess> element in the previous policy or otherwise you will not be able to see the effect of the IF statement during decision making. This is because the <TargetAccess> rules in a PERMIS policy are considered one by one and permission is granted from the moment one of these rules says so.

With the security descriptor given in the previous section, the action arguments actually don't get delivered to the PermisPDP (and hence the PermisPDP sees the action in the decision request as having no arguments). In order to make the action arguments available to the PermisPDP, an additional PIP (the Action PIP) has to be used. The ActionPIP has to be placed before the PermisPDP in the relevant security descriptor. So, the <authz> tag of the security descriptor should read like:

```
<authz value="action:issrg.gt4.pip.ActionPIP
              permisAuthz:issrg.gt4.PermisPDP"/>
```

It is very important that the ActionPIP part comes *before* the PermisPDP! This is because the GT4 framework tackles the PIPs and PDPs in order of occurrence. Next, one has to initialise this PIP (just like the PDP) so in the deployment descriptor file of your service (server-config.wsdd) add a line like

```
<parameter name="action-parameterConfig"
           value="/path/to/parameter/config/file"/>
```

(You should already have a line
<parameter name="permisAuthz-customConfig" value="/path/to/config"/>
in this file)

Note: the parameter name before the dash (i.e. action) has to be the same as the part before the colon in the <authz value line above. Again, you can change this if you want, as long as

the scopes match. On the other hand, the `parameter name` after the dash (i.e. `parameterConfig`) is fixed and cannot be changed.

The contents of the file pointed to by `parameterConfig` must be as follows. For each combination of *service*, *operation* and *argument* this file contains a single line. Each line consists of three parts, separated by spaces. The first part is the end point reference of the service involved, the second part is the action name and the third part is the parameter path. For instance, for the math service the first part could be

https://127.0.0.1:8443/wsrf/services/examples/security/first/MathService

For the add operation, the second part is simply `add`. This is determined by the `<portType>` element in the WSDL file

The third part is:
`{http://www.globus.org/namespaces/examples/MathService_instance_4op}add`

Between the curly brackets you should put the target name space (`tns`) defined in the WSDL file of the service you are trying to protect. The part after the curly brackets is determined from this part of the WSDL file:

```
<message name="AddInputMessage">
    <part name="parameters" element="tns:add"/>
</message>
```

Note that `tns` is defined as the name space included between the curly brackets.

Finally, the `ActionPIP` assumes that the deployment descriptor (`server-config.wsdd`) is at a certain location that is computed from the end point reference. In this case, it will look for the file:
`$GLOBUS_LOCATION/etc/examples_security_first_MathService/server-config.wsdd`

If you don't have this directory and file, simply create the directory and put a symbolic link in it to point to the actual `server-config.wsdd` file.

Putting all of this together, you should now be able to see the effect of using the parameter value in decision making. This is to say, calling the add method with a parameter value bigger than 100 will result in a permission denied, whilst calling it with a value less than 100 will be allowed (provided of course you possess the permisRole = MathServiceUser role).

4 Using VOMS Attributes in Decision Making

4.1 Assumptions

We assume that you know how to generate proxy certificates containing VOMS attributes using `voms-proxy-init`. Information on how to install and configure VOMS client tools for use by the Globus Toolkit may be found on the following websites:

- http://wiki.ngs.ac.uk/index.php?title=Example_VOMS_Client_Installation

This site explains how to install VOMS client tools
- http://wiki.ngs.ac.uk/index.php?title=VOMS_Client_Tools
A wiki page on VOMS Client Tools maintained by the NGS

The use of `voms-proxy-init` is necessary if you want to use the 'push' model. In this scenario the user first collects the VOMS attributes he/she would like to use and these attributes then get pushed to the PERMIS PDP.

If you would like to use the 'pull' model, then you will need access to a SAML Attribute Authoritity that can communicate with a VOMS database.

Note that it is unnecessary to install the VOMS plugins provided by Globus. In fact, doing so will break the use of VOMS attributes with PERMIS authorization. Slightly adapted plugins have been installed when you deployed the `permisAuthzGT4.gar` file.

4.2 The Push Model aka using `voms-proxy-init`

In this model, the user first issues the `voms-proxy-init` command and then submits his/her job (or request). So, in this case the `voms-proxy-init` command replaces the `grid-proxy-init` command. Don't forget to specify any VOs you want to use by using the `-voms` switch. If you don't then `voms-proxy-init` behaves exactly as `grid-proxy-init` and no VOMS attributes will be included in the proxy certificate. As a result VOMS attributes will not be available for decision making. Checking whether VOMS attributes have been included in the proxy certificate is done by doing `voms-proxy-info -all` .

4.2.1 The Security Descriptor

In order to use the VOMS attributes you have to add additional PIPs to the security descriptor. To use the VOMS attributes you have to add the following PIPs: `org.globus.voms.PIP` and `issrg.gt4.voms.VomsPIP`

It is very important that you put the `org.globus.voms.PIP` module *before* the `issrg.gt4.voms.VomsPIP`. module in the authorization chain. This is because the `VomsPIP` module relies on work done by the `org.globus.voms.PIP` module.

The `<authz>` part of your security descriptor could thus read as follows:

```
<authz value="globusvoms:org.globus.voms.PIP
              voms:issrg.gt4.voms.VomsPIP
              permisAuthz:issrg.gt4.PermisPDP"/>
```

This is the minimal authorization chain that will provide PERMIS authorization decisions based on VOMS attributes.

4.2.2 The Deployment Descriptor

The issrg.gt4.voms.VomsPIP needs a configuration file to initialise properly. The location of this configuration file is indicated by the `customConfig` parameter, which is fixed and not configurable. So, in the Web Service Deployment Descriptor for your service (typically `server-config.wsdd`) make sure you add the following line:

```
<parameter name="voms-customConfig"
           value="/path/to/configuration/file"/>
```

Note again how the `voms` part before the dash matches the `voms` part before the colon in security descriptor. (Of course you can choose another identifier here, at long as they match up.)

This configuration file has to provide the necessary information to load the policy. It is highly recommended that this file is the same as used in Section 2. Ensure that the role assignment rules are identical during testing (you can restrict these later for an operational service).

4.2.3 Using VOMS Attributes in the PERMIS Policy

The `VomsPIP` will return the information contained in the VOMS proxy certificate using the following predefined role types:

```
voms_name          ID::= {attributes.109}
voms_group         ID::= {attributes.110}
voms_role          ID::= {attributes.111}
voms_fqan          ID::= {attributes.114}
```

with `attributes=1.2.826.0.1.3344810.1.1`

In particular, these role types should be defined as follows inside the `<RoleHierarchyPolicy>` tag of a PERMIS policy

```
<RoleHierarchyPolicy>
  <RoleSpec Type="voms_name" OID="1.2.826.0.1.3344810.1.1.109">
    <SupRole Value="/issrg.cs.kent.ac.uk"/
    <!-- your vo name here -->
  </RoleSpec>
  <RoleSpec Type="voms_group" OID="1.2.826.0.1.3344810.1.1.110">
    <SupRole Value="/issrg.cs.kent.ac.uk/Kent"/>
    <SupRole Value="/issrg.cs.kent.ac.uk/Kent/cs"/>
     <!-- your vo groups here. You can make them hierarchical if you wish -->
 </RoleSpec>
  <RoleSpec Type="voms_role" OID="1.2.826.0.1.3344810.1.1.111">
    <SupRole Value="/issrg.cs.kent.ac.uk/Kent/cs/Role=ResearchAssociate"/>
    <!-- your vo roles here. Again you can make the roles hierarchical if you wish -->
  </RoleSpec>
  <RoleSpec Type="voms_fqan" OID="1.2.826.0.1.3344810.1.1.114">
    <SupRole Value="/issrg.cs.kent.ac.uk/Kent/Role=NULL/Capability=NULL"/>
    <!-- your vo FQANs here -->
  </RoleSpec>
</RoleHierarchyPolicy>
```

The Policy Editor software is preconfigured to recognise these role types. You only need to specify the values that you will use in your policy.

When the `issrg.gt4.voms.VomsPIP` module is used the resulting attributes are computed from the fully qualified attribute names contained in the VOMS proxy certificate. Suppose your VOMS proxy certificate tells you (through `voms-proxy-info -all`) that you have the attribute:

- `/issrg.cs.kent.ac.uk/Kent/cs/Role=Lecturer/Capability=NULL`

The PERMIS VomsPIP will then assign the following attributes to you:

- `voms_name=/issrg.cs.kent.ac.uk`
- `voms_group=/issrg.cs.kent.ac.uk/Kent`
- `voms_group=/issrg.cs.kent.ac.uk/Kent/cs`
- `voms_role=/issrg.cs.kent.ac.uk/Kent/cs/Role=Lecturer`
- `voms_fqan =/issrg.cs.kent.ac.uk/Kent/cs/Role=Lecturer/Capability=NULL`

Obviously, you will only be able to actually assert these attributes if this is in accordance with the `RoleAssignmentPolicy` of the policy that is being used.

The voms attributes can now be used to give privileges to VO members. For example, say you want to allow members of the /issrg.cs.kent.ac.uk VO to perform additions with the maths service, members of the kent group to perform multiplications and lecturers in the CS subgroup to perform divisions, then your target access policy would be:

```
<TargetAccess>
 <RoleList>
   <Role Type="voms_name" Value="/issrg.cs.kent.ac.uk"/>
 </RoleList>
 <TargetList>
   <Target>
     <TargetDomain ID="MathService"/>
     <AllowedAction ID="add"/>
   </Target>
 </TargetList>
</TargetAccess>
<TargetAccess>
 <RoleList>
   <Role Type="voms_group" Value="/issrg.cs.kent.ac.uk/Kent"/>
 </RoleList>
 <TargetList>
   <Target>
     <TargetDomain ID="MathService"/>
     <AllowedAction ID="multiply"/>
   </Target>
 </TargetList>
</TargetAccess>
<TargetAccess>
 <RoleList>
   <Role Type="voms_role" Value="/issrg.cs.kent.ac.uk/Kent/cs/Role=Lecturer"/>
 </RoleList>
 <TargetList>
   <Target>
     <TargetDomain ID="MathService"/>
     <AllowedAction ID="divide"/>
   </Target>
 </TargetList>
</TargetAccess>
```

4.3 The Pull Model

4.3.1 Configuration

If users want to pull attributes from specified repositories they have to configure the `issrg.gt4.pip.SubjectPIP` Policy Information Point. Again, this has to be added to the security descriptor, like so:

```
<authz value="subject:issrg.gt4.pip.SubjectPIP
              permisAuthz:issrg.gt4.PermisPDP"/>
```

Again, this PIP has to be initialized using a configuration file. The location of this configuration file is specified in the deployment descriptor. The value to be used is again `customConfig`. In the deployment descriptor there thus will be a line reading:

```
<parameter name="subject-customConfig"
           value="/path/to/configuration/file"/>
```

The format of this configuration file is specified in Appendix A, but here we look into more detail at how to specify the SAML Attribute Authority. The SAML AA is specified using the `url` directive from the configuration file. The URL used will contain all the necessary information to connect to the repository.

The protocol to be used is `voms+saml+https`, so a typical URL specification looks like this (all on one line)

```
url=voms+saml+https://<host>:<port>?authn=3;login=8;password=<user's
password>;keystoretype=11;keystorefile=<filepath>;samlport=issrg.saml.
voms.XFireVOMSSAMLPort
```

Possible values for `authn` are:
3: authenticate using a key store
4: authenticate using a key pair
5: authenticate using a proxy certificate
6: authenticate using an encrypted key (pair)

Possible values for `login` are:
7: obtain necessary information interactively
8: obtain necessary information from URL

We now assume that all the information will be obtained from the URL, this is that `login=8` has been specified.

When `authn=3` is used then the following additional information is needed: `password`, `keystorefile` and `keystoretype`.

Possible values for `keystoretype` are:
11: key store file is of type PKCS12
12: key store file is of type JKS

When `authn=4` is used then the following additional information is needed: `keyfile`, `pkcfile` and `keytype`.

Possible values for `keytype` are:
`DSA` and `RSA`

When authn=5 is used then the following additional information is needed:
'proxyfile'. This points to the proxy certificate to use.

When authn=6 is used then the following additional information is needed:
'keyfile', 'pkcfile','keytype' and 'password'.

Possible values for 'keytype' are:
DSA and RSA

The value of `password` should be the password of the encrypted private key.

However when choosing `login=7`, there is no need to specify any of the other information in the URL as you will be prompted for it interactively. However, you should still remember to indicate the authentication method by specifying `authn`.

Also, in both cases (`login=7` and `login=8`) you will need to specify the value of `samlport` which indicates the fully qualified class name of the client to use to contact the SAML Attribute Authority.

At the moment, the only possible value for `samlport` is:
`issrg.saml.voms.XFireVOMSSAMLPort` which is an XFire client with JAXB binding. All the necessary libraries for using this client have been installed when the PERMIS gar file was deployed.

LATER WE WILL DESCRIBE HERE TO SET UP THE TRUSTSTORE

4.3.2 PERMIS Policy

You should check with your SAML Attribute Authority which role type will be used to indicate the various VOMS attributes. The `<RoleHierarchyPolicy>` tag of the PERMIS policy needs to be written such that it is consistent with these returned values.

5 Using SetUID as an Alternative to the Grid Mapfile

5.1 Introduction

The Grid Resource Allocation and Management (GRAM) service provides a single interface for requesting and using remote system resources for the execution of "jobs". The most common use of GRAM is remote job submission and control. It is designed to provide a uniform, flexible interface to job scheduling systems. GT4 contains a GRAM implementation, which uses the Web service interfaces (WS-GRAM). The WS-GRAM grid service can run users jobs under different local user names. The Globus Toolkit grid-mapfile maps 'external' identifiers (PKC DNs) into these local user names. When using grid-map based authorisation, a `UsernamePrincipal` object gets added to the principal list of the client based on the local username specified in the grid-mapfile. This `UsernamePrincipal` object is then used by the WS-GRAM service to run the user's job.

Once an access request is authorised by a configured PERMIS PDP, we can insert a "set user name" obligation into the PERMIS policy which will be returned to GT4. Subsequently this

obligation can be enforced by the `SetUID` module , which adds a `UsernamePrincipal` object to the principal list of the client in the same way as the grid-mapfile PIP.

Using the PERMIS PDP in combination with the `SetUID` obligation object allows us to achieve the same effect as the grid-mapfile, but also provides greater scalability and flexibility. Local username are now mapped to groups of VO users having particular roles or attributes, and conditions can be attached to this mapping. As new users leave and join the VO, the PERMIS policy does not need to be changed, unlike with the grid-mapfile which must be continually updated.

We are going to show you how to achieve this for the `ManagedJobFactoryService`.

The different steps needed to achieve this are the following:

1. Configure the security and deployment descriptors of the service
2. Write the PERMIS policy that protects the service. This policy will have to include obligations.
3. Adapt the `sudoers` file

5.2 Configuring the Security and Deployment Descriptors

The authorization part of the security descriptor should be altered so that it contains the `issrg.gt4.PermisPDP` and `issrg.gt4.SetUID` classes. It is paramount that the `issrg.gt4.PermisPDP` module is called prior to the `issrg.gt4.SetUID` module in the authorization chain. This is achieved by putting the `PermisPDP` class before the `SetUID` class in the security descriptor.

The security descriptor file looks as follows:

```
<?xml version="1.0"?>
<securityConfig xmlns="http://www.globus.org">
    <auth-method>
        <GSITransport/>
        <GSISecureConversation/>
        <GSISecureMessage/>
    </auth-method>
    <authz value="permisAuthz:issrg.gt4.PermisPDP
                  obligation:issrg.gt4.SetUID"/>
</securityConfig>
```

Note that additional PIPs can be put before issrg.gt4.PermisPDP if the application requires this, but this is the minimal authorization chain that gives an alternative to the grid mapfile.

Make sure that the `securityDescriptor` parameter in the deployment descriptor points to this file. Like so:

```
<parameter name="securityDescriptor"
           value="etc/gram-service/permis-security-descriptor.xml"/>
```

This obviously assumes that the security descriptor is located in `$GLOBUS_LOCATION/etc/gram-service/permis-security-descriptor.xml`

The SetUID class doesn't need any configuration, but the PermisPDP class does. This has been explained already in the section about protecting the MathService.

5.3 The PERMIS Policy and the userAccount Obligation

Let's assume we want to allow anyone (i.e. no roles are required) to submit (i.e. all defined actions are permitted) a job to the ManagedJobFactoryService and that we would like the job to run using the local user name dummy.

A <TargetAccess> rule achieving this is the following:

```
<TargetAccess>
  <RoleList/>
  <TargetList>
    <Target>
      <TargetDomain ID="ManagedJobFactoryService"/>
    </Target>
  </TargetList>
  <Obligations>
    <Obligation FulfillOn="Permit"
       ObligationId="http://www.ogf.org/authz/2007/08/oblig/userAccount">
         <AttributeAssignment AttributeId="urn:oid:1.2.826.0.1.3344810.1.1.112"
            DataType="http://www.w3.org/2001/XMLSchema#string">
              dummy
          </AttributeAssignment>
    </Obligation>
  </Obligations>
</TargetAccess>
```

The important part here is the Obligation element; all parts of it should be copied verbatim in your policy except of course the user name to use. This is, you should change dummy to the local user name of your choice.

Note that this obligation can also be created using the Policy Editor, using the Obligations Editor which can be accessed from the "Users's privileges" tab.

A complete policy protecting the createManagedJob method of the ManagedJobFactoryService is given at the end of this section.

So the general idea is that the PERMIS PDP will return access granted but its response will also include the userAccount obligation, which will then be picked up by the SetUID module. Failing to specify the SetUID module will result in the obligation not being processed. In the case of the ManagedJobFactoryService this will result in errors from the GRAM software which will fail to find a suitable user account to run the job under.

5.4 Adapt the sudoers File

None of the above will work if the account that runs the Globus container isn't allowed to sudo into the target account without being asked for authentication. This is why the sudoers file needs adapting.

Editing the suduoers file is done through the visudo command and you will need to be root to do this. It is highly recommended that you don't use any other editor to edit this file.

Add the following two lines to the `sudoers` file:

```
globus ALL=(dummy) NOPASSWD: /usr/local/globus/libexec/globus-gram-local-proxy-tool *
globus ALL=(dummy) NOPASSWD: /usr/local/globus/libexec/globus-job-manager-script.pl *
```

Note: it is important that each line does not contain newline characters. Also, change `dummy` with your own local user account. If multiple user accounts are mentioned in your policy (or policies) then all user accounts need to be mentioned in a comma separated list between the brackets. Furthermore, it is assumed that the Globus container runs as user `globus` . If another account is used then the file should be changed accordingly. Finally, the example assumes that the Globus toolkit is installed in the directory `/usr/local/globus`. We should remark that the `sudoers` file doesn't like symbolic links so one should use the *real* file path.

5.5 Trying it Out

Using `globusrun-ws` it is easy to submit a job. Here we are simply going to `touch` a file in the home directory of user `dummy`. Note that the user issuing `grid-proxy-init` and `globusrun-ws` doesn't have to be user `dummy`. In fact, to better see the intended effect it is best that another account is used to submit the job. That way, you can see that switching the account really has happened.

First, make sure that you have a valid proxy certificate:

```
$$ grid-proxy-init
```

next do (all on a single line)

```
$$ globusrun-ws -submit -F \
https://localhost:8443/wsrf/services/ManagedJobFactoryService \
-c /bin/touch touch_it
```

Obviously, the URL may have to be changed to match your particular installation.

After running this command you should be able to verify that there is indeed a freshly touched file called `touch_it` in the home directory of user `dummy`. This file will also be owned by user `dummy`. This shows that the job has executed under this particular user account.

5.6 Typical Use Case

A typical use case for using the `SetUID` obligation is as follows. For accounting reasons, it may be necessary that users from different VOs run their jobs under different accounts. This is perfectly possible as each `<TargetAccess>` rule can have its own `<Obligations>` element attached to it. In fact, when used as in the above scenario each `<TargetAccess>` rule must have a `userAccount` obligation associated with it; when this is not the case the user will still be authorized but the WS-GRAM software will fail to find a suitable account to run the job under and job submission will ultimately fail.

5.7 Policy Protecting the ManagedJobFactoryService

Here, we give a very simple policy that 'protects' the `ManagedJobFactoryService`. All members of the VOMS /myvo are authorized to use the `createManagedJob` method, but the resulting job will be run as user `dummy`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X.509_PMI_RBAC_Policy>
<X.509_PMI_RBAC_Policy OID="minimalManagedJobFactoryServicePolicy">
<SubjectPolicy>
  <SubjectDomainSpec ID="SubjectDomain">
    <Include LDAPDN=""/>
  </SubjectDomainSpec>
</SubjectPolicy>
<RoleHierarchyPolicy/>
<SOAPolicy>
  <SOASpec ID="TheSOA" LDAPDN=""/>
</SOAPolicy>
<RoleHierarchyPolicy>
  <RoleSpec Type="voms_name" OID="1.2.826.0.1.3344810.1.1.109">
    <SupRole Value="/myvo"/
    <!-- your vo name here -->
  </RoleSpec>
</RoleHierarchyPolicy>
<RoleAssignmentPolicy>
  <RoleAssignment>
    <SubjectDomain ID="SubjectDomain"/>
     <RoleList>
       <Role Type="voms_name" Value="/myvo"/>
     </RoleList>
    <Delegate Depth="0"/>
    <SOA ID="TheSOA"/>
    <Validity/>
  </RoleAssignment>
</RoleAssignmentPolicy>
<TargetPolicy>
  <TargetDomainSpec ID="ManagedJobFactoryService">
    <Include URL="http://localhost/wsrf/services/ManagedJobFactoryService"/>
    <Include URL="https://localhost/wsrf/services/ManagedJobFactoryService"/>
  </TargetDomainSpec>
</TargetPolicy>
<ActionPolicy>
  <Action ID="createManagedJob" Name="createManagedJob"/>
</ActionPolicy>
<TargetAccessPolicy>
  <TargetAccess>
     <RoleList>
       <Role Type="voms_name" Value="/myvo"/>
     </RoleList>
    <TargetList>
      <Target>
        <TargetDomain ID="ManagedJobFactoryService"/>
      </Target>
    </TargetList>
    <Obligations>
      <Obligation FulfillOn="Permit"
        ObligationId="http://www.ogf.org/authz/2007/08/oblig/userAccount">
        <AttributeAssignment AttributeId="urn:oid:1.2.826.0.1.3344810.1.1.112"
          DataType="http://www.w3.org/2001/XMLSchema#string">
           dummy
        </AttributeAssignment>
      </Obligation>
    </Obligations>
  </TargetAccess>
</TargetAccessPolicy>
</X.509_PMI_RBAC_Policy>
```

Appendix A Configuration File

A.1 General Introduction

Here, we give a description of the GT4-Permis configuration file that can be used to specify the information needed to initialize the `PermisPDP` module. This is a text file that is interpreted on a line by line basis. Note that the `PermisPDP` module can also be initialized by configuration parameters as specified in Appendix B.

Comments are possible on a line basis. Any line starting with a '#' character is interpreted as a comment line and is otherwise ignored.

An example of such a file is:

```
ini: xml=/path/to/policy.xml
# previous line tells where the policy is located
#
# tells us which attribute certificate to use. Multiple
# attribute certificates can be specified by repeating this line
...: ac=/path/to/ace/user0.ace
# tells the PERMIS PDP to initialize
...: init
```

This example file contains two directives, namely `xml` and `ac`.

Any configuration file starts with an `ini` command (except when it starts with an `inc` command).

The following directives can appear at most once in a configuration file:

- `soa`
- `oid`
- `xml`
- `acattribute`
- `pkcattribute`
- `init`

The following directives may appear more than once in a configuration file:

- `rootca`
- `url`
- `pkc`
- `ac`

There is an additional command 'inc' that can be used to include additional files. This has the same effect of copying the contents of the target file in the first configuration file. Usage is as follows:

```
inc: path/to/included/file
```

A.2 Meaning of the Directives

There are two different use cases to consider. The first is that the PERMIS policy is stored as an XML file on the local file store. In this case, the argument of the `xml` directive points to this XML file.

The second use case is that the PERMIS policy is stored embedded in a X.509 AC. This X.509 AC can either be stored in a remote repository (typically on an LDAP server) or on the local file store. If it is in a remote repository, the `soa` (Source of Authority) directive gives the LDAP distinguished name of the policy writer. The `oid` directive (Object IDentifier) gives the name (or OID) of the policy to use as stored in the OID attribute of the <X.509_PMI_RBAC_Policy> element of the PERMIS policy. If it is stored on the local file store then it should be pointed to be one of the `ac` directives in the configuration file, but the `soa` and `oid` directives should still be specified.

Thus, either the `xml` directive must be specified (and any `soa` or `oid` directive will be ignored) or both the `soa` and `oid` directive must be specified

The `acattribute` directive specifies the attribute name under which X.509 ACs are stored in remote repositories (pointed to by the `url` directives). If this directive is not specified then a default name ("attributeCertificateAttribute") will be used.

The `pckattribute` directive specifies the attribute name under which PKCs are stored in remote repositories (pointed to by the `url` directives). If this directive is not specified then a default name ("userCertificate") will be used.

The `init` directive denotes the end of the configuration file. At this point all the gathered information will be used to try and construct the PERMIS decision engine.

A `rootca` directive specifies a root of trust for signature verification. It should point to a file on the local file store containing a (self signed) PKC of a root CA (root of trust) ie. this PKC is used as the end point in a signature verification chain. More than one root of trust may be specified.

A `url` directive specifies the URL (or address) of a remote repository to use. This repository may contain user attributes (typically in the form of X.509 ACs, but other forms are possible), user PKCs and also the actual policy (embedded in an X.509 AC) may be stored in a repository. In a typical setup the URLs point to LDAP servers, but also other URLs are possible. For example, one may use a WebDAV repository or a SAML Attribute Authority.

Any `pkc` directive specifies the location on the local file store of a user PKC to use.

Any `ac` directive specifies the location on the local file store of a X.509 AC to use. One can use the `ac` directive in conjunction with `url` directives. Typically the `ac` directive will be used in test scenarios or in use cases where the number of users is not very large. Note that the policy may be stored in such a X.509 AC.

A.3 A Complete Example Configuration File

Here, we give an example of a complete configuration file.

```
ini: soa=cn=A PERMIS Test User,o=PERMISv5,c=GB
...: oid=1.2.826.0.1.3344810.6.0.0.13
```

```
...: url=ldap://sec.cs.kent.ac.uk/
...: acattribute=attributeCertificateAttribute
...: pkcattribute=userCertificate;binary
...: rootca=/usr/local/globus/etc/permisAuthzGT4/rootca.cer
...: init
```

As a matter of fact this configuration could be used to protect the Shutdown service running on localhost inside the GT4 container.

Appendix B Parameters for the `PermisPDP` Module

This section describes the various parameters that are available to initialize the `issrg.gt4.PermisPDP` module. This allows configuring the PermisPDP with the right policy without having to use additional configuration files. All necessary information is contained in the deployment descriptor of the service.

1. The `SOA` parameter gives the name of the policy issuer.
2. The `OID` parameter specifies what policy of that issuer to use. This parameter is necessary because an issuer may have a number of policies for various situations or targets, so the OID parameter is used to distinguish between them. The OID can be any unique string (not just strings in dotted number format) used by a single issuer (or `SOA`). If they are not unique there is no guarantee whatsoever which policy will be used.
3. The `LDAP` parameter tells the URL of the LDAP server that holds the policy.
4. The `LDAP_AC_attribute` parameter specifies the attribute name under which X.509 ACs are stored in the LDAP repository. The default for this is `attributeCertificateAttribute`.
5. The `LDAP_PKC_attribute` parameter specifies the attribute name under which PKCs are stored in the LDAP directory. The default for this is userCertificate.
6. The `ROOT_CA` parameter specifies the location of the PKC of the Root CA that certifies the PKCs of the AC issuers. Note that because Authentication and Authorization are separate domains, this Root CA may be different from the one that signs PKCs for Globus users: this Root CA signs the signature verification keys of the Attribute Authorities that assign roles to the end users.
7. The GSP_useGridShib parameter tells whether the user credentials will be pulled from LDAP (when the parameter is missing or false) or pushed from gridshib. If gridshib is to be used and you wish to use roles stored in ACs then you will also need to specify the name of the Shibboleth attribute that contains role ACs. This is done using the `GSP_AC_LDAP_attribute parameter`.