Table of Contents

The Simplified Policy Language

As already explained here, Argus policies contain collections of rules that state which actions can be performed on which resources by which users. XACML, the language used internally by Argus to define policies, provides great expressiveness and flexibility but it's very hard to read and author for uman beings. For this reason, Argus provides a Simplified Policy Language (SPL) to hide the complexity of XACML while providing much of its flexibility.

As an example, the following policy denies access to all the resources (under Argus control) to the members of the ATLAS VO:

```
resource ".*" {
   action ".*" {
      rule deny { vo = "atlas" }
   }
}
```

The SPL syntax

```
resource <value> {

   action <value> {

      rule <permit|deny> {

         <attributeId>=<attributeValue>
         ...
      }
      ...
   }
   ...
}
...
```

The SPL defines three stanza types: `resource`, `action` and `rule`. It's possible to define multple resouce stanzas that can contain multiple action stanzas that can contain multiple rules stanzas.

The `resource` stanza is used to target a resource (or set of resources, if wildcards are used) under the control of Argus authorization.

The `action` stanza (always defined in the context of an enclosing `resource`) is used to target an action (or set of actions, if wildcards are used) that has to be authorized by Argus on the resource identified by the enclosing `resource` stanza.

The `rule` stanza defines who is authorized (in case of a `permit` rule) or not authorized (in case of a `deny` rule) to perform the action on the resource identified by the enclosing action and resource stanzas.

Identifying actions and resources

Actions and resources are identified by unique identifiers that are assigned to them. This identifiers are usually URIs, but any string that is unique in your deployment may work.

You can also use wildcards in your SPL policies to target group of resources or actions, like in the following example:

```
resource "http://cnaf.infn.it/cream-ce-01" {
```

```
   action ".*" {

      rule permit { vo = "cms" }

   }
}
```

This policy authorizes users from the CMS vo to perform any action on the resource `http://cnaf.infn.it/cream-ce-01`.

Identifying subjects

In Argus policies, the users (or software agents) that need to be authorized to execute an action on a specific resource are identified using a set of attributes, like:

- the subject of the user's X509 certificate;
- the CA that issued the user's x509 certificate;
- the VO the user belongs to;
- whether the use has a specific FQAN in its bag of VOMS attributes;
- whether the user has a specific FQAN as his primary FQAN.

The table below specifies the supported attributes for Argus 1.1:

Attribute name	Description	Example
`subject`	The user's X509 certificate subject in rfc2253 or openssl format	`subject ="CN=Andrea Ceccanti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT`
`subject-issuer`	The subject (in rfc2254 or openssl format) of the CA that issued the user's x509 certificate	`subject-issuer = "CN=INFN CA,O=INFN,C=IT"`
`vo`	The name of the VO the user belongs to	`vo = "atlas"`
`fqan`	The fqan present in the user's bag of VOMS attributes	`fqan="/dteam/Role=VO-Admin"`
`pfqan`	The user primary fqan	`pfqan="/atlas/Role=pilot"`

The contents of the `rule` stanza

As already pointed out, the `rule` stanza defines who is authorized to perform a specific action on a specific resource. A subject can be identified using the attributes defined in the previous section.

```
resource "http://cnaf.infn.it/cream-ce-01" {

   action "submit-pilot-job" {

      rule permit { pfqan="/atlas/Role=pilot" }

   }
}
```

In the above policy, only subjects that have the `/atlas/Role=pilot` fqan as their primary fqan are authorized (since the rule is `permit` rule) to perform the action `submit-pilot-job` on the resource `http://cnaf.infn.it/cream-ce-01`. To prevent users from LHCB VO the execution of the same action, one would write the following policy:

```
resource "http://cnaf.infn.it/cream-ce-01" {

   action "submit-pilot-job" {

      rule deny { vo = "lhcb" }

   }
}
```

Multiple attributes inside the `rule` stanza

It is possibile to define multiple attributes inside a `rule` stanza. All the attributes defined in the rule stanza need to match with the subject attributes present in the authorization request for the rule to be applied. This can be explained more clearly using an example:

```
resource "http://cnaf.infn.it/cream-ce-01" {

   action "submit-job" {

      rule permit {
         vo = "cms"
         subject-issuer = "CN=INFN CA,O=INFN,C=IT"
      }
   }
}
```

The meaning of the above policy is that only members from the VO CMS that have a certificate signed by the `CN=INFN CA,O=INFN,C=IT` CA will be authorized to perform the action `submit-job` on resource `http://cnaf.infn.it/cream-ce-01`. CMS members with certificates signed by the CERN CA, for instance, will not be authorized.

Since all the attributes defined in a rule must be "matched" in the request for the rule to be applied, one can think about multiple attributes inside a rule stanza as conditions that are ANDed to select who will be authorized to perform the action the rule is about.

How policies are evaluated

The first applicable policy (and only that one) that matches the authorization request is the one that is applied by Argus. This means that **order matters**. An example will help in understanding this concept.

Suppose we want to grant access to our CE to all members of VO CMS but not those that have `/cms/Role=pilot` as their primary FQAN. We would write a policy like this:

```
resource "http://cnaf.infn.it/cream-ce-01" {

   action ".*" {

      rule deny{ pfqan = "/cms/Role=pilot"}
      rule permit { vo = "cms" }

   }
}
```

Since the deny rule precedes the permit rule in the above policy, we are able to deny access only to CMS users with the pilot role, but grant access to other members of CMS. This is due to the fact that the first deny rule will not match to CMS users that do not have the pilot role, so the following permit rule will be applied. On the contrary, if we reversed the order of the two rules like in the following policy:

```
resource "http://cnaf.infn.it/cream-ce-01" {

   action ".*" {

      rule permit { vo = "cms" }
      rule deny{ pfqan = "/cms/Role=pilot" }
   }
}
```

the deny rule would be useless, since the permit rule that precedes it would always match any CMS member.

The `obligation` stanza

Starting with Argus version 1.1, the SPL supports `obligation` stanzas. The syntax of the obligation stanza is as follows:

```
obligation "obligationId" {
   [attributeId = attributeValue]*
}
```

Oligation stanzas can be placed either in the resource or action context and are used to define a set operations that must be performed by the Argus PEP in conjuction with an authorization decision. An obligation stanza can define 0..N attribute definitions, that are passed as parameters to the PEP for the fulfillment of the obligation.

An example of policy with an obligation is the following:

```
resource "http://cnaf.infn.it/wn"{

   obligation "http://glite.org/xacml/obligation/local-environment-map" {}

   action "http://glite.org/xacml/action/execute"{
      rule permit { vo = "dteam" }
   }
}
```

The Argus PEP currently supports only the `map-to-local-enviroment` obligation.

The `map-to-local-environment` obligation

The `map-to-local-environment` obligation, identified by the following id:

```
http://glite.org/xacml/obligation/local-environment-map
```

is used within a policy to signify that a mapping to a local posix account will be produced by the Argus server as a result of a permit policy.

The use of this obligation is **mandatory** for the policies that authorize the execution and mapping of pilot jobs on the worker node.

Examples

Ban policies

Ban policies are used to deny a subject on all possible resources. For this reason ban policies need to be placed at the top and defined for any action on all the resources.

```
resource ".*" {
    action ".*" {
        rule deny { subject = "CN=Alberto Forti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT" }
        rule deny { fqan = /dteam/test }
    }
}
```

Glexec on the WN policies

Policy that authorize execution and mapping of pilot jobs on the WN need to specify the `map-to-local-environment` obligation to produce a mapping that gLexec can use to do the user switch. An example of such policy is the following:

```
resource "http://cnaf.infn.it/wn"{

   obligation "http://glite.org/xacml/obligation/local-environment-map" {}

   action "http://glite.org/xacml/action/execute"{
      rule permit { vo = "dteam" }
      rule permit { pfqan = "/atlas/Role=pilot" }
      rule permit { pfqan = "/ops/Role=pilot" }
   }
}
```

The above policy authorizes the execution of jobs on the WN by:

- people from the dteam VO,
- people that have `/atlas/Role=pilot` as the primary fqan
- people that have `/ops/Role=pilot` as the primary fqan

This topic: EGEE > SimplifiedPolicyLanguage
Topic revision: r12 - 2010-06-15 - unknown