

---

# **Argus Documentation**

***Release 1.6***

**Andrea Ceccanti, Valery Tschopp, Michel Jouvin**

February 28, 2016



<b>1</b>	<b>Argus Authorization Service</b>	<b>3</b>
1.1	Summary . . . . .	3
1.2	Argus Service Installation . . . . .	4
1.3	Service Components . . . . .	4
1.4	Enabled Applications . . . . .	5
1.5	Support and Monitoring . . . . .	5
1.6	Development Information . . . . .	5
1.7	Additional Support . . . . .	7
1.8	About the name Argus . . . . .	7
<b>2</b>	<b>Argus Concepts</b>	<b>9</b>
2.1	Argus Introduction . . . . .	9
2.2	Attribute Based Descriptions . . . . .	9
2.3	Identifiers within Argus . . . . .	12
2.4	Resource ID . . . . .	12
2.5	Conclusion . . . . .	13
<b>3</b>	<b>Argus: Policy Administration Point (PAP)</b>	<b>15</b>
3.1	Argus Policy Administration Point (PAP) Installation . . . . .	15
3.2	Argus Policy Administration Point (PAP): Configuration . . . . .	15
3.3	Argus: Policy Administration Point (PAP): Operation . . . . .	19
3.4	Argus Policy Administration Point (PAP): Administration . . . . .	19
3.5	The Simplified Policy Language . . . . .	30
3.6	Argus: Policy Administration Point (PAP): Known Issues . . . . .	34
<b>4</b>	<b>Argus: Policy Decision Point (PDP)</b>	<b>35</b>
4.1	Argus Policy Decision Point (PDP) Installation . . . . .	35
4.2	Argus Policy Decision Point (PDP): Configuration . . . . .	35
4.3	Argus Policy Decision Point (PDP): Operation . . . . .	39
4.4	Argus: Policy Decision Point (PDP): Troubleshooting . . . . .	41
<b>5</b>	<b>Argus: Policy Enforcement Point Daemon (PEP)</b>	<b>43</b>
5.1	Argus PEP Server Installation . . . . .	43
5.2	Argus PEP Server: Configuration . . . . .	43
5.3	Argus PEP Server: Operation . . . . .	47
5.4	Argus PEP Server: Troubleshooting . . . . .	49
5.5	Argus PEP Server Policy Information Points (PIP) . . . . .	49
5.6	Argus PEP Clients . . . . .	61

5.7	Argus PEP Client Library: C API . . . . .	63
5.8	Argus PEP Client: Java Programming Interface . . . . .	64
5.9	Argus GSI PEP Callout . . . . .	66
5.10	Argus PEP Server Obligation Handlers . . . . .	70
<b>6</b>	<b>Argus Monitoring</b>	<b>77</b>
6.1	Nagios Probes for Argus (UMD) . . . . .	77
<b>7</b>	<b>Example of Authorization Requests and Policies</b>	<b>85</b>
7.1	User Based Authorization . . . . .	85
7.2	Per-VO Pilot Job Authorization Policy . . . . .	86
<b>8</b>	<b>Legacy Pages from Twiki site</b>	<b>89</b>
8.1	Argus EMIR Publisher Configuration (EMI-3) . . . . .	89
8.2	Argus Services (EMI-1) Fine Tuning . . . . .	90
8.3	Argus Development Tools . . . . .	92
8.4	GGUS User Support for Argus . . . . .	93
8.5	Authorization Service: Load and Lifetime Testing . . . . .	94
8.6	Authorization Services Testing Summary . . . . .	97
8.7	Authorization Services Testing Summary . . . . .	99
8.8	Authorization Services Testing Summary . . . . .	101
8.9	Argus: Quick Start: Manual Installation . . . . .	102
8.10	Argus: Quick Start: Site Policy Setup . . . . .	103
8.11	Argus: Quick Start: Manual Configuration . . . . .	106
8.12	Quick Start: glite 3.2 Argus Installation . . . . .	106
<b>9</b>	<b>Argus Service Deployment for EMI</b>	<b>109</b>
9.1	Requirements . . . . .	109
9.2	Installation with YUM . . . . .	109
9.3	Update with YUM . . . . .	110
9.4	Configuration with YAIM . . . . .	110
<b>10</b>	<b>Nagios Probes for Argus</b>	<b>113</b>
<b>11</b>	<b>EMIR Publisher for Argus 1.6 (EMI-3)</b>	<b>115</b>
<b>12</b>	<b>Known Issues</b>	<b>117</b>
12.1	Timeouts for certificates from CAs that use OCSP . . . . .	117
12.2	Performance issue with Argus PEP Server (EMI-2, EMI-3, all versions) . . . . .	117
12.3	Problem the EMI-3 update and Argus PEP Server v.1.6.1 . . . . .	118
12.4	Problem with Argus 1.6 (EMI-3) and <code>fetch-crl</code> . . . . .	118
12.5	Problem with Nagios plugins for Argus and TMP directory permission . . . . .	119
12.6	Problem with Argus 1.5 (EMI-2) and CREAM . . . . .	119
12.7	Problem with upgrade from Argus 1.4 (EMI-1) to Argus 1.5 (EMI-2) . . . . .	119
<b>13</b>	<b>Authorization Service: Grid Map File Syntax</b>	<b>121</b>
13.1	Description . . . . .	121
13.2	File Syntax . . . . .	121
13.3	Subject Keys . . . . .	121
13.4	Map Targets . . . . .	122
<b>14</b>	<b>Argus YAIM Configuration for EMI</b>	<b>123</b>
14.1	YAIM Configuration for ARGUS_server . . . . .	123
<b>15</b>	<b>Indices and tables</b>	<b>125</b>



Contents:



## Argus Authorization Service

### 1.1 Summary

The Argus Authorization Service renders consistent authorization decisions for distributed services (e.g., user interfaces, portals, computing elements, storage elements). The service is based on the XACML standard, and uses authorization policies to determine if a user is allowed or denied to perform a certain action on a particular service.

The Argus Authorization Service is composed of three main components:

- The Policy Administration Point (PAP) provides the tools to author authorization policies, organize them in the local repository and configure policy distribution among remote PAPs.
- The Policy Decision Point (PDP) implements the authorization engine, and is responsible for the evaluation of the authorization requests against the XACML policies retrieved from the PAP.
- The Policy Enforcement Point Server (PEP Server) ensures the integrity and consistency of the authorization requests received from the PEP clients. Lightweight PEP client libraries are also provided to ease the integration and interoperability with other EMI services or components.

The following graphic shows the interaction between the components of the service:

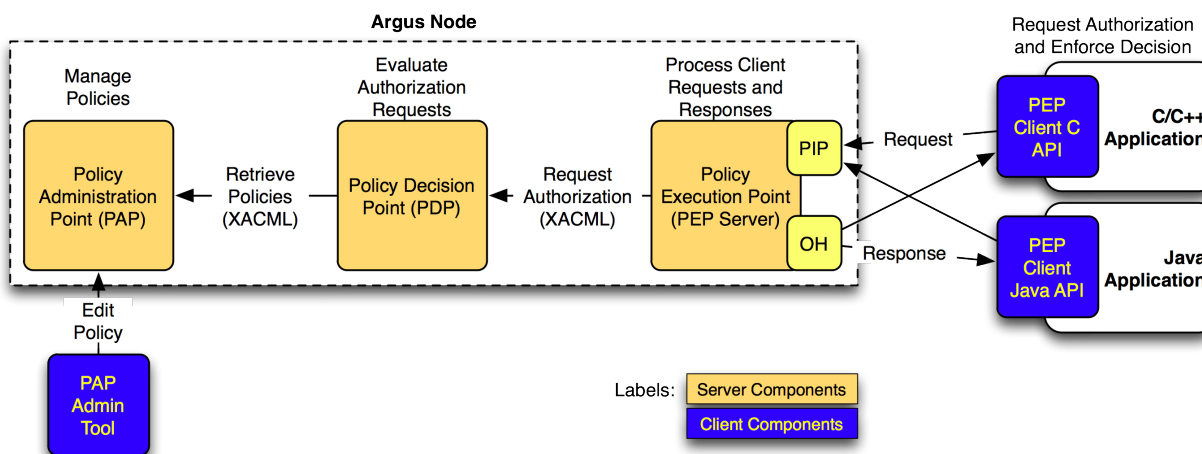


Fig. 1.1: Argus service components

**Note:** In Argus, the PEP is separated in a client/server architecture. The PEP Server handles the lightweight PEP client requests, and runs on the Argus node.

## 1.2 Argus Service Installation

The following section provides instructions for setting up an Argus environment quickly. It does not provide an exhaustive description of every possible deployment model or configuration option, that can be found in the following *Service Components* and *Enabled Applications* sections.

Before you continue it is recommend that you read this *introduction to the Argus system*. This will provide you with a better understanding of how the components work together, what information passes between the components and how policies are formed.

### 1.2.1 Argus EMI Deployment

For EMI, the Argus Service is installed with YUM, and configured with YAIM. Please follow the Argus *Deployment for EMI* documentation.

### 1.2.2 gLExec Worker Node with Argus Deployment

To install and configure an Argus compatible gLExec worker node, follow these *GLExec Argus Quick Installation Guide*

## 1.3 Service Components

If you are beginning to install the authorization service from scratch, you should install the components in the order listed here; PAP, then PDP, then PEPd. You don't have to, but it makes the most sense for most use cases.

### 1.3.1 PAP: Policy Administration Point

The *Policy Administration Point* (PAP) provides three major functions:

- Provide the tools for authoring policies
- Store and manage authored policies
- Provide managed policies to other authorization service components

### 1.3.2 PDP: Policy Decision Point

The *Policy Decision Point* (PDP) is a policy evaluation engine. The PDP receives authorization requests from Policy Enforcement Points and evaluates these requests against authorization policies retrieved from the PAP.

<a href="#">Installation</a>	<a href="#">Configuration</a>	<a href="#">Operation</a>	<a href="#">Troubleshooting</a>
------------------------------	-------------------------------	---------------------------	---------------------------------

### 1.3.3 PEP: Policy Enforcement Point

The *Policy Enforcement Point* (PEP) is the client to the authorization service. It gathers information relevant to an authorization request (e.g. who the user, what action they are attempting to perform, which service they are attempting to perform the action on, etc.) and sends the request to the PDP for evaluation. The PEP then acts upon returned result by allowing the request to proceed (in the case a positive authorization decision) or by denying the action (in the event of a negative decision).

In Argus, the PEP itself has a client/server architecture



- The PEP Server handles the lightweight PEP client requests, and runs on the Argus node. See the following links for more information on the PEP Server: [Installation](#), [Configuration](#), [Operation](#), [Troubleshooting](#)
- Lightweight PEP client libraries are available to authorize requests from the application side, and to enforce decision locally. There are two variants available:
  - PEP Client C API: [Programming Interface \(API\)](#)
  - PEP Client Java API: [Programming Interface \(API\)](#)

*PEP command-line clients* are also available

## 1.4 Enabled Applications

The following applications contain an Argus PEP client and can make authorization requests to the Argus service.

Table 1.1: Argus-enabled Applications

GSI PEP Callout	<a href="#">Introduction</a>	<a href="#">Installation</a>	<a href="#">Configuration</a>	<a href="#">Troubleshooting</a>
gLExec with PEP Plugin	<a href="#">Introduction</a>	<a href="#">Installation</a>	<a href="#">Configuration</a>	<a href="#">Troubleshooting</a>

## 1.5 Support and Monitoring

### 1.5.1 GGUS Support

General support (installation, site administrator) for Argus is available through [GGUS](#)

### 1.5.2 Argus Support Mailing List

Argus specific (developer, site administrator) questions can be sent directly to the [argus-support@googlegroups.com](mailto:argus-support@googlegroups.com) mailing list. You don't need a Google email address or a Google account to send or receive emails from this mailing list.

- **Subscription:** To subscribe to the support mailing list, simply send an email to: [argus-support+subscribe@googlegroups.com](mailto:argus-support+subscribe@googlegroups.com)
- **Unsubscribe:** You can unsubscribe from the list at anytime by sending an email to: [argus-support+unsubscribe@googlegroups.com](mailto:argus-support+unsubscribe@googlegroups.com)

**NOTE:** The mailing list was previously [argus-support@cern.ch](mailto:argus-support@cern.ch), but it have been migrated to [argus-support@googlegroups.com](mailto:argus-support@googlegroups.com) at the end of the EMI project (April 2013).

### 1.5.3 Nagios Monitoring

Nagios plugins are available to monitor an Argus server.

- **EMI:** [EMI Argus Nagios Probes Documentation](#)

## 1.6 Development Information

Argus is an open-source product hosted on [GitHub](#). In addition to this user-oriented documentation, you can find a description of Argus architecture and main components at Argus [main site](#).

## 1.6.1 Specifications

- XACML 2.0 Specifications eXtensible Access Control Markup Language (XACML) Version 2.0
- SAML 2.0 Profile of XACML, Version 2 (Working Draft 5)
- XACML Profile for the gLite WN XACML Grid Worker Node Authorization Profile (v. 1.0)
- XACML Profile for the gLite CE XACML Grid Computing Element Authorization Profile (v. 1.0)
- XACML Authorization Profile for EMI Common XACML Authorization Profile (v. 1.1)

## 1.6.2 Requirements

- Requirements Document (*EDMS document 944192*)
- Testing Plan (*EDMS document 986067*)

## 1.6.3 Presentations

- *20110531-EMI\_AllHands\_2011-Argus\_Integration.ppt*: Argus - EMI Authorization Integration (EMI AH 2011, 31 May 2011, Lund)
- *20110412-EGI\_UF\_2011-Argus.ppt*: Argus - EMI Authorization Service (EGI UF 2011, 12 April 2011, Vilnius)
- *20100917-EGI-TF\_ArgusSecurity.ppt*: Argus Security (EGI TF 2010 Security Session, 17 Sept. 2010, Amsterdam)
- *introduction\_authz\_service.ppt*: General introduction to the authorization service
- *100602\_argus\_intro\_rod.ppt*: Introduction to Argus for ROD (EGI ROD Workshop, 2 June 2010, Amsterdam)

## 1.6.4 Source Code Information

We have migrated the Argus source code to [GitHub](#).

- Argus Authorization Service: <https://github.com/argus-authz>

The source code was previously stored in the CERN subversion server. **Please do not use the SVN repository anymore**

- WebSVN View: <https://svnweb.cern.ch/world/wsvn/glxa>

## 1.6.5 Development Tools

The Argus PT uses the following development tools.

For performance and load testing we use the following testing suite.

## 1.6.6 Argus Production Settings and Optimization

Production sites can optimize the Argus Service settings to their specific needs. Please have a look at the Argus Fine Tuning documentation.

### 1.6.7 Performance and Load Testing

Results and metrics of the performance and load testing can be found here:

- For a physical machine:
  - Versions 1.3.0 and 1.3.1
  - Version 1.4.0
- For a virtual machine:
  - Version 1.4.0

## 1.7 Additional Support

- [HERAS-AF](#) project has supported the project by providing a good XACML policy engine and excellent, ongoing, support of their code.
- YourKit is kindly supporting this open source projects with its full-featured Java Profiler. YourKit, LLC is the creator of innovative and intelligent tools for profiling Java and .NET applications. Take a look at YourKit's leading software products: [YourKit Java Profiler](#) and [YourKit .NET Profiler](#).
- This product includes software developed by the [Caucho Technology](#).

## 1.8 About the name Argus

In Greek mythology Argus was a 100-eyed giant that was meant to watch and protect various things and people including the Goddess Io. He was slain by Hermes but the gods chose to preserve his hundred eyes and affix them to the tail-feathers of a brilliantly colored bird, the peacock, in homage. The peacock logo is provided by the royalty free clip art site [clker.com](#).



---

## Argus Concepts

---

### 2.1 Argus Introduction

The Argus authorization service is designed to answer questions in the form of *Can user X perform action Y on resource Z at this time?* Not surprisingly, two pieces of information are required to answer this question; the request that describes X, Y, and Z and the policy against which the request is evaluated. The purpose of this introduction is to provide an understanding of the logical contents of the request and the policy. Such an understanding will help in creating appropriate access control policies for a service. This introduction does not cover the command line tools, simple policy language, or underlying XACML policies used by the authorization service. That information can be found in the Policy Administration Point documentation.

### 2.2 Attribute Based Descriptions

Before discussing the request and response it is important to understand how users, actions, and resources are identified. Most grid deployers will be familiar with X.509 subject distinguished names (DNs) based access control. Access control systems which render authorization decisions based only on an identifier, like a subject DN, are known as identity-based access control systems. However, an application usually knows more about a user than simply their identifier. It may know for which organization the user works, or groups and professional organizations to which the user belongs. An access control system which can render a decision based on a set of attributes describing the users/resources/actions is known as an attribute-based access control system. Note, an attribute-based access control system where policies are only based on an identifier attribute is functionally equivalent to an identity-based access control system.

Argus is an attribute-based system. It uses attributes to identify the user attempting to perform an action, the resource on which the action is to be performed, the action itself, and other environmental information. Within the system an attribute is made of 4 pieces of information: a unique identifier/name, one or more values for the attribute, the data type of the attribute values (e.g. a string, an integer, an email address), and an optional identifier for the issuer of the attribute. The following are examples of attribute based descriptions.

In an identity based system a user (more properly known as a subject) might have the identity 'jsmith'. In an attribute based system the subject might be identified by the following attributes:

- id: subject-id, datatype: string, value: jsmith
- id: org, datatype: string, value: CERN
- id: affiliation, datatype: string, value: employee
- id: vo, datatype: string, value: lhcb, cms, atlas

In another example an action in an identity based access control system might be identified by the string 'submit-job'. In an attribute based system the action might be identified with the following attributes:

- `id: action-id, datatype: string, value: submit-job`
- `id: pilot-job, datatype: boolean, value: false`
- `id: executable, datatype: string, value: /usr/bin/myexec`
- `id: expected-execution-duration, datatype: integer, value: 10`

The exact attributes available within a request are determined by the application seeking an authorization decision. Obviously each unique application will have different actions it wishes to protect, a different means of describing itself, and possibly different information about the subject. Also, as can be seen by these examples, the amount of information upon which a decision could be based is larger within an attribute based access control system than an identity based system. As demonstrated, the single piece of identity data used in an identity based system can also be used within an attribute based one.

### 2.2.1 The Request

An authorization decision request is simply a set of four different attributes collections. The four collections are:

- *subject* - contains attribute describing user who is trying to perform an action
- *action* - contains attributes describing the action the subject is attempting to perform
- *resource* - contains attributes describing the program within which the subject is attempting to perform the action
- *environment* - contains attributes relevant to the decision but not part of the previous three collection (e.g. time/date of the request, machine used by the subject, etc.).

A request must contain at least one attribute in each of the *subject*, *action*, and *resource* sets but it is common for there to be no environment attributes.

### 2.2.2 The Policy

A policy is a collection of rules that are evaluated to determine the result of an authorization request. The result of a policy evaluation may be:

- *permit* - indicates the subject is permitted to perform the action on the resource
- *deny* - indicates the subject is not permitted to perform the action on the resource
- *not applicable* - indicates no policy applied to the request and so no decision could be reached
- *indeterminate* - indicates there was an error evaluating the policy

In order to determine whether a rule is met each rule contains a *target*. The target defines one or more combination of attribute which trigger the rule. When more than one combination is listed any one may trigger the rule. For example, the target of a rule may stipulate the following combination:

- subject attribute `vo` has a value of *atlas* and the `id` of the action is *job-submit* and the `id` of the resource is *cern-ce*
- subject attribute `id` has a value of *john* and the `id` of the action is *job-submit* and the `id` of the resource is *cern-ce*
- subject attribute `id` has a value of *jane* and the `id` of the action is *job-submit* and the `id` of the resource is *cern-ce*

As you can see, each combination of attributes within the target needs to be complete (that is they need to describe the whole situation that would trigger a rule), there is no mechanism for expressing logical operations like AND, OR, NOT. As you can imagine this could lead to a lot of repeated information. To help alleviate this, policies, the container of rules, can also contain targets. Therefore the previous rule target could be written as follows:

- A policy with target: the `id` of the action is *job-submit* and the `id` of the resource is *cern-ce*
- A `permit` rule with the target combination:
  - subject attribute `vo` has a value of *atlas*
  - subject attribute `id` has a value of *john*
  - subject attribute `id` has a value of *jane*

You can also have multiple rules within a single policy, for example:

- A policy with target: the `id` of the action is *job-submit* and the `id` of the resource is *cern-ce*
- A `permit` rule with the target combinations:
  - subject attribute `vo` has a value of *atlas*
  - subject attribute `id` has a value of *john*
  - subject attribute `id` has a value of *jane*
- A `deny` rule with the target combinations:
  - subject attribute `id` has a value of *christoph*

In fact, policies can even contain other policies which helps in the case where you may want a policy about a resource which in turn contains a policy about actions (at that resource) and finally rules for that action. For example:

- A policy with target: `id` of the resource is *cern-ce* contains:
  - A policy with target: the `id` of the action is *job-submit*
  - A `permit` rule with the target combination:
    - \* subject attribute `vo` has a value of *atlas*
  - A `deny` rule with the target combination:
    - \* subject attribute `id` has a value of *christoph*
  - A policy with target: the `id` of the action is *job-manage*
  - A `permit` rule with the target combination:
    - \* subject attribute `id` has a value of *john*
    - \* subject attribute `id` has a value of *jane*

### 2.2.3 Not Applicable and Indeterminate Results

In some cases, either because no policy applied to the request or because there was an internal error evaluating a policy, a decision other than `permit` or `deny` is returned. In the event that a result of `not applicable` or `indeterminate` is returned Argus will treat this as a `deny`. Thus Argus is said to be a *deny biased* system.

### 2.2.4 Example Requests and Policies

Look at the *dedicated page* [<argus\\_request\\_and\\_policy\\_examples>](#) for request and policy examples.

## 2.3 Identifiers within Argus

Argus makes use of various identifiers for attribute identifiers and values. The identifiers must be URIs (URNs or URLs). The responsibility for defining such identifiers falls to different individuals, depending on the expected scope of use of the identifiers.

### 2.3.1 Entity ID

For those familiar with other grid services one of the first differences that you will encounter when setting up Argus is the requirement to provide *entity IDs* for the service components. These IDs provide a way of uniquely identify a logical service component. We use the term “logical service component” because each service component may be clustered. So the logical instance is the set containing all the physical instances participating in the cluster.

It is the deployer of service component that determines this identifier. The identifier may be any URI for which the deployer is authoritative. That is, it must come from a domain (in the case of a URL) or namespace (in the case of a URN) that the deployer controls. The following formula is a reasonable means for generating these identifiers: `http://{authz_domain}/{service_component_identifier}`. The domain should be a domain name that is not linked to any particular servers hostname but is instead related to the service (e.g. `authz.example.org`). The component identifier should be `pap`, `pdp`, or `pepd` depending on which component is being installed.

If an organization runs more than one component an additional qualifier may be added to appended to the path of the URL identifier. For example an organization that runs a different PEPd for each computer cluster they operate might choose the entity IDs `http://authz.example.org/pepd/cluster1` and `http://authz.example.org/pepd/cluster2`

## 2.4 Resource ID

The resource identifier used to identify the resource being protected by the authorization service (e.g. a compute cluster, a portal). This identifier is specific to a given instance of the resource. If an organization runs two different portals each portal receives a different ID. These identifiers may then be used within a policy in order to indicate policies that apply to the specific resource (i.e. policy A applies to portal 1 and policy B applies to portal 2).

The resource identifier is selected by the deployer of the resource. Two reasonable ways for generating this identifier are:

- if the resource has a natural URI identifier associated with it (e.g. the main page of the portal) that may be used
- if the resource does not have a natural URI identifier a synthetic one may be created with the following formula: `http://{authz_domain}/resource/{resource_identifier}` The `authz_domain` should be a domain name that is not linked to any particular server’s hostname but is instead related to the authorization service (e.g. `authz.example.org`). The resource identifier is simply a unique string for the protected resource. A human intelligible string is best (e.g. `sequencingPortal`).

### 2.4.1 Action ID

The action identifier is used to identify the action for which the resource is requesting authorization. This action identifier is specific to a given piece of software but all instances of that application use the same identifier for a given action, it is not deployment specific.

The action identifier is selected by the developer of the application. The identifier may be generated in one of two ways:



- If the application is widely used a request could be made to a standards body, such as [IANA](#), for an unique identifier for each action of the application. This way all implementations of the specification will likely use the same identifier and thus make it a bit easier for policy writers.
- The application developer may generate an identifier. A reasonable formula for such an identifier is: `http://{application_domain_name}/{application_name}/action/{action_id}` where the domain name is the domain component of the application's website and the action ID is a human readable string for the action (e.g. `=readFile,addUser`)

## 2.4.2 Attribute ID

The attributes produced by an application identify bits of information that the application was able to gather and make available for an authorization request. The identifiers are specific to a given piece of software but all instances of that application use the same identifier for a given attribute, it is not deployment specific.

The attribute identifier is selected by the developer of the application. The identifier may be generated in a couple of ways:

- If the attribute is already defined by an existing standard (the LDAP schema standards are a great place to start looking) the identifier from that standard may be used. In the case of the LDAP schema the URN `urn:oid:{ldap_attribute_oid}` can be used.
- If the attribute is likely to be widely used a request could be made to a standards body, such as [IANA](#), for an unique identifier for this attribute. This way all implementations of the specification will likely use the same identifier and thus make it a bit easier for policy writers.
- The application developer may generate an identifier. A reasonable formula for such an identifier is: `http://{application_domain_name}/{application_name}/attribute/{attribute_id}` where the domain name is the domain component of the application's website and the attribute ID is a human readable string for the attribute (e.g. `username,entitlements`)

In general, application developers should prefer already defined attributes over creating their own.

## 2.5 Conclusion

At this point you should understand what an attribute is, that a request is made of subject, action, resource, and environment attributes, what a rule is and that policies are a collection of rules. You should also understand that a policy or rule is triggered if any one of the combination of attribute/values listed within its target is present within the request.



---

## Argus: Policy Administration Point (PAP)

---

The Policy Administration Point (PAP) provides three major functions:

- Provide the tools for authoring policies
- Store and manage authored policies
- Provide managed policies to other authorization service components

### 3.1 Argus Policy Administration Point (PAP) Installation

To deploy the Argus PAP for EMI, Please follow the documentation *Argus Deployment for EMI*.

### 3.2 Argus Policy Administration Point (PAP): Configuration

#### 3.2.1 Configuration Files

The PAP is configured through the use of two files: `pap_configuration.ini` and `pap_authorization.ini`, located in the `/etc/argus/pap` (EMI), or `$PAP_HOME/conf` (gLite), directory. Most of the information contained in these files can also be set through the command line interface (which is the **recommended** way to do configuration on the PAP).

#### 3.2.2 Service configuration file

The service is primarily configured through the `pap_configuration.ini` configuration file. This file is a standard INI file with five defined sections.

##### Section: [paps]

This section contains configuration about paps. The information in this section should be set via the *PAP CLI*.

A paps can be defined by providing the following information (the R value in the Required? column indicates information that is required only for remote paps):

Property	Description	Re-quired?	Default Value
<i>alias.type</i>	Defines a pap as <code>local</code> or <code>remote</code> .	Y	None
<i>alias.public</i>	Visibility of the pap: <code>true</code> or <code>false</code> . If false its policies are not sent to other paps.	N	false
<i>alias.enabled</i>	<code>true</code> or <code>false</code> . If false its policies are not sent to PDPs.	N	false
<i>alias.dn</i>	DN of the PAP to get policies from.	R	None
<i>alias.hostname</i>	Hostname of the PAP to get policies from.	R	None
<i>alias.port</i>	Port of the PAP to get policies from.	N	8150
<i>alias.path</i>	Path of the services exposed by the PAP to get policies from.	N	/pap/services
<i>alias.protocol</i>	Protocol to use to contact the remote PAP.	N	https

### Section: [paps:properties]

This section contains information about policy distribution and pap ordering.

Property	Description	Re-quired?	Default Value
<i>poll_interval</i>	The polling interval (in seconds) for retrieving remote policies.	Y	None. Recommended value is 14400 (4 hours).
<i>ordering</i>	Comma separated list of pap aliases. Example: alias-1, alias-2, ..., alias-n. Defines the order of evaluation of the policies of the paps, that means that the policies of pap “alias-1” are evaluated for first, then the policies of pap “alias-2” and so on.	N	If not specified the default pap is always the first one.

### Section: [repository]

This section contains information about the PAP policy repository.

Property	Description	Re-quired?	Default Value
<i>location</i>	Path to the repository directory.	N	\$PAP_HOME/repository
<i>consistency_check</i>	Forces a consistency check of the repository at startup.	N	false
<i>consistency_check.repair</i>	if set to <code>true</code> automatically fixes problems detected by the consistency check (usually means deleting the corrupted policies).	N	false

### Section: [standalone-service]

This section contains information about the PAP standalone service.

Property	Description	Required?	Default Value
host-name	The hostname or IP address the service will bind to	N	127.0.0.1
port	The service port number	N	8150
shutdown_port	The service shutdown port number	N	8151
shutdown_command	The command string that must be received on the shutdown port in order to shutdown the service. The command is needed in order to prevent unauthorized shutdown commands coming from localhost. This is effective only if the pap_configuration.ini file is not world-readable. If the option is not present in configuration, no check on the command will be made.	N	shutdown
entity_id	This is a unique identifier for the PAP. It must be a URI (URL or URN). If a URL is used it need not resolve to any specific webpage.	N	The service endpoint, e.g. <code>https://pap.cern.ch:8150/pap/services/</code>

### Section: [security]

This sections contains information about PAP security configuration.

Property	Description	Required?	Default Value
certificate	The X.509 pem-encoded service certificate	Y	/etc/grid-security/hostcert.pem
private_key	The unencrypted private key bound to the certificate	Y	/etc/grid-security/hostkey.pem
trust_store	The directory where CA files and CRL are looked for	N	/etc/grid-security/certificates
crl_update_interval	How frequently the PAP should update CRLs, CAs and namespaces from the filesystem. The interval is defined as a string with the following format: N{s,m,h,d} where N is the number of either (s=seconds, m=minutes, h=hours, d=days).	N	30m

### 3.2.3 Service Access Control

Access control rules are configured through the `pap_authorization.ini` configuration file. Authorization is based on off of the Subject DN or VOMS attribute within the client certificate used to authenticate to the PAP.

#### PAP permissions

The authorization layer is based on an Access Control List (ACL), composed of several Access Control Entries (ACEs). Each ACE defines the actions that an administrator is allowed to execute on the PAP. Administrators' privileges are defined in terms of PAP permission flags, whose meaning is described in the table below:

Permission Flag	Meaning
POLICY_READ_LOCAL	Allows read access to locally defined policies
POLICY_READ_REMOTE	Allows read access to policies imported from remote PAPs
POLICY_WRITE	Allows write access to locally defined policies
CONFIGURATION_READ	Allows read access to PAP configuration
CONFIGURATION_WRITE	Allows write access to PAP configuration
ALL	All of the above permissions

A set of permission flags can be assigned to an administrator by defining an ACE in the `pap_authorization.ini` configuration file or by using the authorization management commands provided by the `pap-admin` command line interface.

ACEs are expressed as

```
<principal> : <permission>
```

couples.

The principal part of the ACE is either:

- ANYONE, to assign privileges to any authenticated user (i.e., any user that presents a trusted certificate).
- a VOMS FQAN, e.g., `/atlas/Role=VO-Admin`
- a **quoted** X509 certificate subject, e.g., `"/C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Andrea Ceccanti"`

The permission part of the ACE is either:

- a single PAP permission flag, e.g. `CONFIGURATION_READ`
- a | separated list of PAP permission flags, e.g. `POLICY_READ_LOCAL|CONFIGURATION_READ`, to grant a set of permissions.

So, for example, to grant `POLICY_READ_LOCAL` and `POLICY_READ_REMOTE` permissions to a user identified by an x509 certificate with `/C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Andrea Ceccanti` subject, one should write:

```
"/C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Andrea Ceccanti" : POLICY_READ_LOCAL|POLICY_READ_REMOTE
```

Note that the subject has been put into quotes! For VOMS FQANs this is not needed (FQAN syntax does not allow whitespaces inside the FQAN), so one could write:

```
/atlas/Role=PAP-Admin : ALL
```

Authorization entries are loaded at PAP service startup time so any `pap_authorization.ini` modifications done “by hand” while the PAP service is running do not take effect until the PAP service is restarted.

To modify the PAP authorization configuration at runtime, use the authorization management commands provided by the `pap-admin` CLI. Changes made to the PAP ACL by these commands are immediately reflected on the `pap_authorization.ini` file.

## Configuration File Syntax

In the `pap_authorization.ini` file, ACEs are grouped in two *stanzas* according to the type of the principal. Currently, two stanzas are supported:

- `[dn]`, that lists ACEs defined for principals identified by an X509 certificate subject.
- `[fqan]`, that lists ACEs defined for principals identified by VOMS fqans.

An example of configuration file is given below:

```
[dn]
"/C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Andrea Ceccanti" : ALL
ANYONE : CONFIGURATION_READ|CONFIGURATION_WRITE
[fqan]
/voms-ws/Role=PAP-Admin : ALL
```

## 3.3 Argus: Policy Administration Point (PAP): Operation

### 3.3.1 Service Operation Commands

The service can then be started using the `PAP_HOME/bin/pap-standalone start` command.

The service can be stopped using the `PAP_HOME/bin/pap-standalone stop` command.

### 3.3.2 Service Ports

- Default standalone service port: 8150
- Default standalone shutdown service and status port: 8151

### 3.3.3 Service Endpoints

- `/pap/services/ProvisioningService` : this endpoint provides the policy provisioning interface
- `/shutdown` : this endpoint instructs the PAP standalone service to shutdown and is reachable only from localhost on the configured shutdown and status port
- `/status` : this endpoint provides current status information on the PAP is reachable only from localhost on the configured shutdown and status port

All the web services implemented by the PAP can be reached at the `/pap/services/` context.

### 3.3.4 Service Logs

The PAP standalone log file (`pap-standalone.log`) can be found in the `$PAP_HOME/logs` directory. If the PAP is deployed on top of tomcat, the log file (`pap.log`) can be found in the `$CATALINA_HOME/logs` directory.

## 3.4 Argus Policy Administration Point (PAP): Administration

The CLI (Command Line Interface) allows to perform all of the policy management operations as well as to set most of the configuration information of the PAP including authorization settings. All these operations are accessible by means of (sub)commands of the PAP CLI and they are divided in three sections: *policy management*, *pap management* and *authorization management*.

The command to invoke the CLI is `pap-admin`.

### 3.4.1 Running the pap-admin client

Type:

```
pap-admin --help
```

to get a list of the commands supported by the current version of the PAP command line client.

The general usage is the following:

```
usage: pap-admin [global-options] <command> [options] [args]
```

- `global-options` : are options shared by all commands (all these options are reported by `--help`).
- `command` : the command to perform.
- `options` : command specific options.

Type:

```
pap-admin <command> --help
```

to see the available command specific options.

Global options:

- `-h, --help`: help message;
- `-v, --verbose`: set verbose commands output;
- `--host <arg>`: specifies the target PAP hostname (default is localhost). This option defines the PAP endpoint to be contacted as follows: <https://arg:port/pap/services/>;
- `--port <arg>`: specifies the port on which the target PAP is listening (default is 8150);
- `--url <arg>`: specifies the full target PAP endpoint (default: <https://localhost:8150/pap/services/>);
- `--cert <arg>`: specifies non-standard user certificate;
- `--key <arg>`: specifies non-standard user private key;
- `--password <arg>`: specifies the password used to decrypt the user's private key;
- `--proxy <arg>`: specifies a user proxy to be used for authentication;

By default, the `pap-admin` is configured to contact a PAP daemon on localhost port 8150 (i.e., the standalone PAP daemon). If you want to contact a PAP on another host use the `--host` option, a PAP listening on a different port use the `--port` option. Alternatively the full endpoint can be specified with the `--url` option.

**NOTE:** The Argus PDP caches the policy pulled from the PAP. If you make policy changes to the PAP you will need to restart the PDP in order to force it to refresh its policy. Also, the PEPd caches results from the PDP for a short time (up to 10 minutes, by default) so you may need to restart the PEPd as well.

#### Exit status

Each command returns 0 for success or non-zero in case of error.

**X509 authentication** A valid X509 certificate or proxy certificate is needed in order to run the `pap-admin` client. The certificate to be used by the command is found as follows:

1. if the user is `root` then the client looks for host certificate at the usual location, i.e. `/etc/grid-security/hostcert.pem` and `/etc/grid-security/hostkey.pem`. These lo-



cations can be overridden by setting the `X509_USER_CERT` and `X509_USER_KEY` environment variables. 2. if the user is not `root`, then the client looks for a proxy certificate at the usual location, i.e., `/tmp/x509up_u%GREEN%<uid>%ENDCOLOR%`. The default proxy location can be overridden using the `X509_USER_PROXY` environment variable. If no proxy is found, then the client looks for a certificate (and the relative private key) at the usual location, i.e., `$HOME/.globus/usercert.pem` and `$HOME/.globus/userkey.pem`. These locations can be overridden by setting the `X509_USER_CERT` and `X509_USER_KEY` environment variables.

This behaviour can be overridden by specifying the `--cert`, `--key` and `--proxy` command line options.

### Client configuration (since version 1.3.0)

The `pap-admin.properties` file, found in `/etc/argus/pap` (EMI) or `/opt/argus/pap/conf` (gLite), allows to set the following properties:

Property	Description	Default Value
host	The PAP host that will be contacted by the <code>pap-admin</code> CLI if no host is explicitly specified with the <code>--host</code> option	localhost
port	The remote PAP service port number	8150

This property file is especially useful when administering a PAP installed on a remote machine.

## 3.4.2 Policy Management Commands

This set of commands allows to perform policy management operations. All the commands in this section modifies the *default* pap. In order to target another local pap use the option `--pap <alias>`.

### Command: list-policies

List policies. By default the policies of the *default* pap are listed unless option `--pap` is specified.

```
usage: pap-admin [global-options] list-policies [options]
```

Command specific options:

- `--pap <alias>`: list policies of pap “alias” ( *default* pap is assumed if this option is missing);
- `--all`: list policies of all the defined paps;
- `-srai, --show-ra-ids`: show resource and action ids;
- `-sai, --show-all-ids`: show all ids (resource, action and rule ids);
- `--show-xacml`: print policies using XACML.

### Command: ban

Allows to ban an attribute (i.e. SUBJECT, FQAN, CA, etc.).

A *deny* rule is added for the given attribute into the specified resource/action value. If the resource or the action values are not specified then `”*.*"` is assumed.

```
usage: pap-admin [global-options] ban [options] <id> <value>
```

- `id`: id of the attribute. The list of supported id depends on the Argus version:

- Argus v. 1.0: dn, ca, vo, fqan, pfqan.
- Argus v. 1.1: subject, subject-issuer, vo, fqan, pfqan.
- **value:** value of the attribute - **note:** If you are using Argus v. 1.0, the DN must be in RFC2253 format, which can be obtained from openssl using the command `openssl x509 -in <cert.pem> -noout -subject -nameopt rfc2253`

Command specific options:

- `-a, --action <value>`: specify an action value (default is `".*"`)
- `-r, --resource <value>`: specify a resource value (default is `".*"`)
- `--pap <alias>`: add the policy to the pap “alias” ( *default* pap is assumed if this option is missing)
- `--private`: set the policy as private
- `--public`: set the policy as public

**Semantic of the command:** the *resource* and the *action* where the deny rule is inserted are chosen as follows:

- if the first resource found in the repository matches the given one, then that resource is used, otherwise a new one is created.
- if a matching resource was found, then if its first action matches the given one then this action is used, otherwise a new action is created (i.e. inside the new resource or inside the matched resource).
- otherwise a new resource/action are created and inserted on the top.

Example:

```
pap-admin ban subject "CN=host.test.foo.it, L=FOO, OU=Host, O=ORGANIZATION, C=IT"
```

### Command: un-ban

Allows to un-ban an attribute (i.e. SUBJECT, FQAN, CA, etc.), that means removing a deny rule (if it exists in the given resource/action) for the given attribute.

```
usage: pap-admin [global-options] un-ban [options] <id> <value>
```

- **id:** id of the attribute. The list of supported id depends on the Argus version:
  - Argus v. 1.0: dn, ca, vo, fqan, pfqan.
  - Argus v. 1.1: subject, subject-issuer, vo, fqan, pfqan.
- **value:** value of the attribute - **note:** If you are using Argus v. 1.0, the DN must be in RFC2253 format, which can be obtained from openssl using the command `openssl x509 -in <cert.pem> -noout -subject -nameopt rfc2253`

Command specific options:

- `-a, --action <value>`: specify an action value (default is `".*"`)
- `-r, --resource <value>`: specify a resource value (default is `".*"`)
- `--pap <alias>`: remove the ban policy from the pap *alias* ( *default* pap is assumed if this option is missing)

**Semantic of the command:** the target *resource* and *action* to search the deny rule for are chosen as follows:

- the target *resource* is the first matching *resource* in the repository;
- inside the target *resource* the target *action* is the first matching *action* ;
- if no target *resource* or *action* were found than the result is an error message saying “ban policy not found”.

Example:

```
pap-admin un-ban subject "CN=host.test.foo.it, L=FOO, OU=Host, O=ORGANIZATION, C=IT"
```

## Command: add-policy

Add a permit/deny policy.

```
usage: pap-admin [global-options] add-policy [options] <permit|deny> <id=value>...
```

- `permit|deny`: effect of the policy.
- `id=value`: a string in the form “<id>=<value>”, where *id* is any of the attribute ids that can be specified in the simplified policy language and *value* the value to be assigned (e.g. `fqn=vo/group`).

Required command options: 1 `--action-id <action-id>` optionally with `--rule-id <rule-id>`: allows to specify an action-id to insert the policy into. 1 `--resource <value>` and `--action <value>`: allows to specify a resource/action value to insert the policy into.

The two groups (1 and 2) of required options are mutually exclusive.

This command allows to add a (permit/deny) rule into an action by specifying an action-id (in this case the action must already exist) or a resource/action value. In the latter case a new resource and/or action are created if they don't already exist. The command returns an error if there are more than one existing resource and/or action with the same value. By default the rule is inserted at the top of an action unless the `--bottom` option is given. If the `--rule-id` is set the rule is inserted before the given rule-id or after if the `--after` option is present.

Command specific options:

- `--pap <alias>`: add the policy to the pap “alias” ( *default* pap is assumed if this option is missing);
- `--action-id <action-id>`: specify an action id;
- `--rule-id <rule-id>`: specify a rule id (requires option `--action-id`);
- `--resource <value>`: specify a resource value;
- `--action <value>`: specify an action value;
- `--after`: insert the rule after the given rule id;
- `--bottom`: insert the rule at the bottom of the list of rules of the action.
- `--obligation <obligationId>`: specify an obligation. (**since version 1.2.0**)
- `--obligation-scope <scope>`: Defines in which scope the obligation will be defined. Possible values: `action`, `resource`. If not specified, `resource` is used as default. (**since version 1.2.0**)

## Command: add-policies-from-file

Add policies (resources or actions) defined in the given file.

```
usage: pap-admin [global-options] add-policies-from-file [options] <file> [resourceId]
```

- `file`: text file containing the policies to add (policies defined with the simplified policy language)
- `resourceId`: the *resource* to insert the policies into.

If *resourceId* is not specified then *file* must contain *resource* elements that will be added, by default, at the bottom (unless option `--pivot` is specified). Otherwise if *resourceId* is not specified then *file* must contain *action* elements that will be added, by default, at the bottom inside *resourceId* (unless option `--pivot` is specified).

Command specific options:

- `--pap <alias>`: add the policies to the pap “alias” ( *default* pap is assumed if this option is missing);
- `--pivot <id>`: insert before <id>;
- `--after`: modifies the behavior of the `--pivot` option in insert after <id>.

### Command: update-policy-from-file

Update a resource/action with a new resource/action defined in a given file.

```
usage: pap-admin [global-options] update-policy-from-file [options] <id> <file>
```

- `id`: id, as listed by the command `pap-admin lp --show-all-ids` command, of the resource or action to be updated;
- `file`: text file containing the new policy definition (using the simplified policy language syntax).

In order to modify an action the `file` must contain only the new action, for example:

```
action ".*" {  
    rule deny { subject="/DC=ch/DC=cern/OU=Organic Units/OU=Users/CN=user/CN=111111/CN=user name" }  
}
```

Command specific options:

- `--pap <alias>`: update the policies for pap “alias” ( *default* pap is assumed if this option is missing);

### Command: remove-policy

Remove policy by id.

```
usage: pap-admin [global-options] remove-policy [options] id...
```

- `id`: id, as listed by the command `pap-admin lp --show-all-ids` command, of the policy (resource, action or rule) to remove;

Command specific options:

- `--pap <alias>`: remove policies of pap “alias” ( *default* pap is assumed if this option is missing);

### Command: remove-all-policies

Remove all policies of a pap. Use option `--pap` to specify a pap different than the default one.

```
usage: pap-admin [global-options] remove-all-policies [options]
```

Command specific options:

- `--pap <alias>`: remove the policies of pap “alias” ( *default* pap is assumed if this option is missing);

### Command: move

Move a resource, action or rule before or after another, respectively, resource, action or rule.

```
usage: pap-admin [global-options] move [options] <id> <pivotId>
```

- `id`: id, as listed by the command `pap-admin lp --show-all-ids` command, of the policy (resource, action or rule) to move;
- `pivotId`: id of the pivot policy ( *id* is moved before *pivotId* )

If *id* refers to a resource, action or rule then *pivotId* must be, respectively, a resource, action or rule id.

Command specific options:

- `--pap <alias>`: move the policy of pap “alias” ( *default* pap is assumed if this option is missing);
- `--after`: move *id* after *pivotId*.

### Command: add-obligation (since version 1.2.0)

Adds on obligation to an existing resource or action policy.

```
usage: pap-admin [global-options] add-obligation <policyId> <obligationId>
```

- `policyId`: the id of the policy where the obligation is to be added. In order to get the `policyId` of existing policies, run the `list-policies` command with the `--show-all-ids` option.
- `obligationId`: the id of the obligation that will be added.

Command specific options:

- `--pap <alias>`: add on policies defined in the pap “alias” ( *default* pap is assumed if this option is missing);

### Command: remove-obligation (since version 1.2.0)

Removes an obligation from an existing resource or action policy.

```
usage: pap-admin [global-options] remove-obligation <policyId> <obligationId>
```

- `policyId`: the id of the policy where the obligation is to be removed. In order to get the `policyId` of existing policies, run the `list-policies` command with the `--show-all-ids` option.
- `obligationId`: the id of the obligation that will be removed.

Command specific options:

- `--pap <alias>`: add on policies defined in the pap “alias” ( *default* pap is assumed if this option is missing);

## 3.4.3 PAP Management Commands

This set of commands allows to perform management operations of the PAPs.

### Command: ping

Ping a PAP and return version information.

```
usage: pap-admin [global-options] ping
```

## Command: add-pap

Add a remote or local pap.

```
usage: pap-admin [global-options] add-pap [options] <alias> [<endpoint> <dn>]
```

- *alias*: a friendly (unique) name used to identify the pap
- *endpoint*: endpoint of the remote pap in the form: `<verbatim>[<protocol>://]<host>[:<port>/[path]]</verbatim>`
- *dn*: DN of the remote pap

A just added pap is disabled by default (its policies are not sent to the PDP), use the command *enable-pap* to enable it.

By default a pap is considered to be private (use the `--public` option to set the pap as public). Policies defined in a public pap can be fetched from other remote PAPs, while this is not allowed when the PAP is set to private.

If *endpoint* and *dn* are present the pap is considered to be remote (unless option `--local` is specified), otherwise it is local. For the endpoint the only required parameter is the hostname, these are the default values:

- *protocol*: `https`
- *port*: `8150`
- *service path*: `pap/services`

When a new pap is added, the PAP service tries immediately to fetch its policies. If the remote pap is not reachable, the `pap-admin` command prints an error message clarifying that the pap was successfully added, but the fetching of the policies failed.

If the option `--no-policies` is given, the policies are not fetched at pap creation time but automatically by the server every polling interval seconds or manually when the `refresh-cache` command is sent to the server.

Examples of endpoint are:

- `test.site.com` (hostname);
- `test.site.com:9999` (hostname and port);
- `test.site.com:9999/service_path` (hostname, port, and service path);
- `https://test.site.com:9999/service_path` (full URL).

Command specific options:

- `-l, --local`: set the pap as local;
- `--remote`: set the pap as remote;
- `--private`: set the pap as private;
- `--public`: set the pap as public;
- `--no-policies`: do not fetch the policies now.

Example:

```
pap-admin add-pap cnaf_pap test.cnaf.infn.it "/C=IT/O=INFN/OU=Host/L=CNAF/CN=test.cnaf.infn.it"
```

## Command: update-pap

Update pap information.

```
usage: pap-admin [global-options] update-pap [options] <alias> [<endpoint> <dn>]
```

The input is the same as for the “add-pap” command, the effect is to update old information with the new one. The *alias* of a pap cannot be modified. In the case of a remote pap the policies are fetched immediately unless option `--no-policies` is given.

### Command: remove-pap

Remove a pap and delete its policies.

```
usage: pap-admin [global-options] remove-pap <alias>
```

- `alias`: alias of the pap to remove

### Command: list-paps

List all defined paps.

```
usage: pap-admin [global-options] list-paps [options]
```

Command specific options:

- `-l`: use a long list format (displays all the information of a pap).

### Command: enable-pap

Set a pap as enabled (i.e. PDPs will get its policies).

```
usage: pap-admin [global-options] enable-pap <alias>
```

### Command: disable-pap

Set a pap as disabled (i.e. PDPs won't get its policies).

```
usage: pap-admin [global-options] disable-pap <alias>
```

### Command: get-paps-order

Get paps ordering.

```
usage: pap-admin [global-options] get-paps-order
```

If no ordering is defined the output message is: No ordering has been defined. If the *default* pap is not listed in the ordering (like in the no ordering defined case) by default it is placed for first.

### Command: set-paps-order

Define paps ordering.

```
usage: pap-admin [global-options] set-paps-order [alias]...
```

- `alias`: a valid pap alias.

All the aliases must be valid (existing). If no arguments are given then the current ordering (if there's any defined) is deleted.

Example:

The remote pap *osct* contains banning policies and we want that policies to be evaluated for first. This is command to issue:

```
pap-admin set-paps-order osct default
```

If the PAP service contains other paps beyond the *osct*, then their policies are evaluated after the *osct* and *default* pap policies. Since the ordering contains only the *osct* and the *default* paps it is not guaranteed a special order for the evaluation of the policies of all the other paps (except that they are evaluated after these two paps).

### Command: refresh-cache

Invalidates the local policy cache and retrieves policies from remote paps.

```
usage: pap-admin [global-options] refresh-cache [alias]...
```

- *alias*: a valid pap alias.

The arguments identify the paps that will be contacted. If no arguments are given, all the defined remote paps are contacted.

### Command: get-polling-interval

Get the polling interval in seconds.

```
usage: pap-admin [global-options] get-polling-interval
```

### Command: set-polling-interval

Invalidates the local policy cache and retrieves policies from remote paps.

```
usage: pap-admin [global-options] set-polling-interval <seconds>
```

- *seconds*: polling interval in seconds.

## 3.4.4 Authorization Management Commands

This set of commands implement Access Control List (ACL) management for PAP administrators.

### Command: list-acl

The `list-acl` command provides an easy way of knowing the authorization configuration of a running PAP.

Typing:

```
pap-admin list-acl
```



prints out the Access Control Entries (ACEs) comprising the ACL currently defined for the running PAP.

Example:

```
~# pap-admin list-acl

/voms-ws/Role=PAP-Admin :
    POLICY_READ_LOCAL|POLICY_READ_REMOTE|POLICY_WRITE|CONFIGURATION_READ|CONFIGURATION_WRITE

"/C=IT/O=INFN/OU=Personal Certificate/L=CNAF/CN=Andrea Ceccanti" :
    POLICY_READ_LOCAL|POLICY_READ_REMOTE|POLICY_WRITE|CONFIGURATION_READ|CONFIGURATION_WRITE

ANYONE :
    CONFIGURATION_READ|CONFIGURATION_WRITE
```

Required permissions : CONFIGURATION\_READ.

### Command: add-ace

The `add-ace` command allows to add (or change) an ACE to the PAP ACL. Note that if an ACE entry already exists on the server for the principal specified in the command, the permissions in such ACE are **replaced** by the ones specified in the command.

Usage:

```
pap-admin add-ace <principal> <permissions>
```

where:

- `principal` can be either an X509 DN or a VOMS FQAN. `ANYONE` can be used to assign permissions to any authenticated user.
- `permissions` is a | separated list of *PAP permissions* that will be assigned to `principal`. The `ALL` shortcut can be used to assign all permission.

Example:

```
pap add-ace '/atlas/Role=VO-Admin' 'ALL'
```

Required permissions: CONFIGURATION\_READ | CONFIGURATION\_WRITE

### Command: remove-ace

The `remove-ace` command removes an ACE from the PAP ACL.

Usage:

```
pap-admin remove-ace <principal>
```

where:

- `principal` can be either an X509 DN or a VOMS FQAN. `ANYONE` can be used to remove permissions assigned to any authenticated user.

Example:

```
pap remove-ace '/atlas/Role=VO-Admin'
```

Required permissions: CONFIGURATION\_READ | CONFIGURATION\_WRITE

## 3.5 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 human 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" }
  }
}
```

### 3.5.1 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 multiple resource 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.

### 3.5.2 Identifying actions and resources

Actions and resources are identified by unique identifiers that are assigned to them. These 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://cnafe.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://cnafe.infn.it/cream-ce-01`.

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

### 3.5.4 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://cnafe.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://cnafe.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://cnafe.infn.it/cream-ce-01" {  
  
    action "submit-pilot-job" {  
  
        rule deny { vo = "lhcb" }  
  
    }  
  
}
```

### Multiple attributes inside the rule stanza

It is possible 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://cnafe.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://cnafe.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.

## 3.5.5 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://cnafe.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://cnaif.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.

### 3.5.6 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]*
}
```

Obligation 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 conjunction 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://cnaif.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-environment 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.

### 3.5.7 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

## 3.6 Argus: Policy Administration Point (PAP): Kown Issues

There is no know issues.

---

## Argus: Policy Decision Point (PDP)

---

The Policy Decision Point (PDP) is a policy evaluation engine.

The PDP receives authorization requests from Policy Enforcement Points and evaluates these requests against authorization policies retrieved from the PAP.

### 4.1 Argus Policy Decision Point (PDP) Installation

To deploy the Argus PDP for EMI, Please follow the documentation *Argus Deployment for EMI*.

### 4.2 Argus Policy Decision Point (PDP): Configuration

#### 4.2.1 Configuration File Syntax

The PDP is configured through the use of the `pdp.ini` file. This file is a standard INI file with three defined sections. The `SERVICE` section contains properties related the PDP service as a whole and how it listens for incoming requests. The `POLICY` section contains properties for the retrieval of policies from the Policy Administration Point (PAP). The final section, `SECURITY`, contains properties that related to various security aspects of the service, the services private key and certificate, for example.

## 4.2.2 Basic Configuration Options

### SERVICE section

Property	Description	Required?	Default Value
entityID	This is a unique identifier for the PDP. It must be a URI (URL or URN) and the same entity ID should be used for all PDP instances that make up a single logical PDP. If a URL is used it need not resolve to any specific webpage.	Y	None. Recommended value is a URL corresponding to the logical PDP service (e.g. <code>http://pdp.example.org</code> ).
hostname	This is the hostname or IP address to which the service will bind.	Y	None.
port	This is the port to which the service will bind.	N	8152
admin-Host	The hostname upon which the service will listen for admin commands.	N	127.0.0.1
admin-Port	This is the port upon which the service will listen for admin command.	N	8153
admin-Password	This is the password required to accompany admin commands. If unspecified than no password is required to run admin commands.	N	None

### POLICY section

Property	Description	Required?	Default Value
paps	A space separated list of PAP endpoint URLs. Endpoints will be tried in turn until one returns a successful response. This provides limited failover support. If more intelligent failover is necessary or load balancing is required, a dedicated load-balancer/failover appliance should be used.	Y	None
retentionInterval	The number of minutes the PDP will retain (cache) a policy retrieved from the PAP. After this time is passed the PDP will again call out to the PAP and retrieve the policy.	N	240 (4 hours)



## SECURITY section

Property	Description	Re- quired	De- fault Value
servicePrivateKey	An absolute path to the file containing the unencrypted, PEM-encoded, private key used by this service. All PDPs instances within a single logical PDP should use the same key	Yes	None
serviceCertificate	An absolute path to the file containing the unencrypted, PEM-encoded, certificate used by this service. All PDPs instances within a single logical PDP should use the same key	Yes	None
trustInfoDir	An absolute path to the directory that contains standard X.509 trust information, such as the IGTF Trust Anchor Distribution	Yes	None
enableSSL	Indicates whether the service port should use SSL/TLS or not	No	false
require-ClientCertAuthentication	Indicates whether the client must use a valid client certificate to authenticate to the PDP	No	false

## Example pdp.ini files

The following example file contain the bare minimum required for a valid PDP configuration file.

```
[SERVER]
entityID = http://argus.example.org/pdp
hostname = argus.example.org

[POLICY]
paps = https://argus.example.org:8150/pap/services/ProvisioningService

[SECURITY]
servicePrivateKey = /etc/grid-security/hostkey.pem
serviceCertificate = /etc/grid-security/hostcert.pem
trustInfoDir = /etc/grid-security/certificates
# HTTPS enabled
enableSSL = true
```

## 4.2.3 Advanced Configuration Options

The following advanced options are available but are unlikely to ever be used by deployers. They are mostly for performing very fine-grained tuning of request/response handling parameters. Incorrectly configuring these can have a very negative impact on performance so deployers should not change these unless they are very sure they understand what the impact will be.

**SERVICE section**

Property	Description	Required?	Default Value
maximumRequests	The maximum number of requests that will be processed simultaneously. Additional requests will be queued.	N	200
requestQueueSize	The maximum number of requests that will be queued up when all the processing threads are busy. Incoming requests received when all processing threads are busy and the queue is full will receive an HTTP 503 error.	N	500
connectionTimeout	This is the length of time, in seconds, the service will wait for the client to send information before it considers the request timed out.	N	30 seconds
receiveBufferSize	This is the size, in bytes, that will be allocated to the HTTP request buffer.	N	16384 (16kb)
sendBufferSize	This is the size, in bytes, that will be allocated to the HTTP response buffer.	N	16384 (16kb)

**POLICY section**

Property	Description	Required	Default Value
policy-SetId	The ID of the policy to fetch from the PAP	No	-1
connectionTimeout	This is the length of time, in seconds, the PAP client will wait for the PAP to send information before it considers the request timed out	None	30
receive-BufferSize	This is the size, in bytes, that will be allocated to the PAP client send buffer	No	16384 (16KB)
send-BufferSize	This is the size, in bytes, that will be allocated to the PAP client request buffer	No	16384 (16KB)

**SECURITY section**

Property	Description	Required?	Default Value
trustInfoRefresh	The frequency, in minutes, that the trust material specified by <code>trustInfoDir</code> will be checked for updates.	N	60 (1 hour)
messageValidityPeriod	The number of seconds, from the time a message is issued, until it is considered expired.	N	300s (5 minutes)
clockSkew	The allowance, in seconds, used when computing validity periods.	N	30s

## 4.3 Argus Policy Decision Point (PDP): Operation

### 4.3.1 Service Operation Commands

#### EMI - Argus 1.3

```
/etc/init.d/argus-pdp start Start the PDP service
/etc/init.d/argus-pdp stop Stop the PDP service
/etc/init.d/argus-pdp restart Restart the PDP service
/etc/init.d/argus-pdp status Provides PDP service status information
/etc/init.d/argus-pdp reloadpolicy Causes the currently cached copy of the policies received from
the PAP can be flushed from memory, and retrieved anew from the PAP
```

#### gLite 3.2 - Argus 1.2

```
$PDP_HOME/sbin/pdpctl.sh start starts the service
$PDP_HOME/sbin/pdpctl.sh stop stops the services
$PDP_HOME/sbin/pdpctl.sh status provides service status data
$PDP_HOME/sbin/pdpctl.sh reloadPolicy causes the currently cached copy of the policy received from
the PAP can be flushed from memory, and retrieved anew from the PAP
```

### 4.3.2 Service Information

#### Service Ports

- Default Service Port: 8152
- Default Admin Port: 8153

The PDP service only requires the standard service port to be open to those PEPd services which will communicate with the PDP. The PDP must also be able to make outbound connections to those PAPs from which remote policies will be retrieved.

#### Service Endpoint URLs

This service contains the following endpoint URLs:

- `https://HOSTNAME:8152/authz` - This endpoint is the recipient of authorization requests.
- `http://127.0.0.1:8153/status` - This endpoint provides current status information on the PDP. This endpoint is password protected.
- `http://127.0.0.1:8153/reloadPolicy` - This endpoint instructs the PDP to flush, and retrieve anew, its policy from the PAP. This endpoint is password protected.
- `http://127.0.0.1:8153/shutdown` - This endpoint instructs the PDP to shutdown. This endpoint is password protected.

**Note:** Admin services may be password protected and thus not invocable without this password.

### 4.3.3 Logging and Logs

This service uses the [logback](#) logging library. Java developers are probably familiar with Apache Log4J, logback is written by the developer who initially wrote Log4J and contains a cleaner API and is much more performant. The configuration file for the logging system can be found in `$PDP_HOME/conf/logging.xml` and changes to this file are picked up every 5 minutes.

#### Enable Debug Logging

To enable debug logging follow: 1 Locate the line that contains `logger name="org.glite.authz"` (line 10 in the default logging config) 1 On the following line, change `INFO` to `DEBUG`

In some cases it may be helpful to see the policy being evaluated for each request. To do this: 1 Locate the line that contains `org.glite.authz.message.policy` (line 22 in the default logging config) 1 On the following line, change `INFO` to `DEBUG`

In some cases it may be helpful to see the incoming and outgoing messages. To do this: 1 Locate the line that contains `org.glite.authz.message.protocol` (line 27 in the default logging config) 1 On the following line, change `INFO` to `DEBUG`

**NOTE** always change your logging levels back to their original values once you are done debugging a problem. Keeping the system on the debug logging level could fill up your disk partition in a short time.

#### Service Logs

The service writes three different logs, located in `/var/log/argus/pdp` (EMI) or `$PDP_HOME/logs` (gLite):

- **process.log** - This log file contains the normal, human-oriented logging messages that the system generates while in operation. This can be thought of as the debug log (though that's really only true if configured to log debug messages).
- **access.log** - This file is an Apache-style access log showing information about incoming requests. This log is meant to be machine parsed. Each line contains the following 'I' (pipe) delimited fields:
  - request time - time of the request, in the UTC timezone, from the Unix epoch (Jan 1, 1970, 00:00:00 UTC)
  - remote host - the hostname or IP address of the host that contacted the PDP
  - server host - the hostname or IP address on which the server received the request
  - server port - the port on which the server received the request
  - request path - the URL path that was requested
- **audit.log** - This file contains information useful for auditing the system (e.g. results of authorization requests). This log is meant to be machine parsed. Each line contains the following 'I' (pipe) delimited fields:
  - request time - time of the request, in the UTC timezone, from the Unix epoch (Jan 1, 1970, 00:00:00 UTC)
  - requester ID - the entity ID of the PEP that made the request
  - request ID - the ID of the incoming authorization request
  - policy ID - the ID of the policy set that was evaluated in order to reach the authorization decision
  - policy version - the version of the policy that was evaluated in order to reach the authorization decision
  - policy decision - the authorization decision that was reached
  - response ID - the ID of the authorization response sent back to the PEP

## 4.4 Argus: Policy Decision Point (PDP): Troubleshooting

### 4.4.1 PDP Uses “Old” Policies

The PDP caches policies received from the PAP in order to avoid the cost of fetching and parsing them for every request. In cases where you know, or suspect, the policy used by the PDP is no longer in synch with the policy stored at the PAP you may use the `pdctl reloadPolicy` command to force the PDP to flush its policy cache and retrieve the latest policy from the PAP.

### 4.4.2 Private Key File Access

Many systems protect their private keys so that only super-user accounts can read them. Starting, and running the PDP, as such an account is strongly discouraged. The recommend approach is to create a special group (e.g. ‘hostkey’) that has read permissions to the key and ensure the user running the PDP service is also in this group. This group should **not** have write permission to the key.

Some people might view this as a loss of security, because, if the service user account were compromised the attacker would be able to read the private key. However, the service holds a copy of key in memory once it starts and this copy can easily be accessed via tools that come with the JRE.



---

## Argus: Policy Enforcement Point Daemon (PEP)

---

In the Argus authorization framework the Policy Enforcement Point is implemented as a client-server architecture. The PEP Daemon is the server component and there are different lightweight PEP clients available: C client, Java client, GSI PEP Callout client, ...

### 5.1 Argus PEP Server Installation

#### 5.1.1 EMI Installation

To deploy the Argus PEP Server for EMI, Please follow the documentation *Argus Deployment for EMI*

### 5.2 Argus PEP Server: Configuration

#### 5.2.1 Configuration File

The PEP is configured through the user of the `pepd.ini` file. This file is a standard INI file with different defined sections. The `SERVICE` section contains properties related the PEP Server service as a whole and how it listens for incoming requests. The `PDP` section that control how callouts to the PDP are made. The final section, `SECURITY`, contains properties that related to various security aspects of the service, the services private key and certificate, for example.

#### PIP Configuration

Each Policy Information Point (PIP) is configured in its own INI section, referenced in the `SERVICE` section under the `pips` property.

The configuration properties needed by each PIP is specific to that given PIP. Some will not require any properties beyond the standard set while other will need addition information.

- See the *Policy Information Point* documentation for more information:
  - *Request Validator PIP*
  - *OpenSSL Subject Converter PIP*
  - *gLite Grid Authorization Profile PIP*
  - *Common XACML Authorization Profile PIP*

## Obligation Handler Configuration

Each *Obligation Handler* (OH) is configured in its own INI section, referenced in the `SERVICE` section under the `obligationHandlers` property.

The configuration properties needed by each Obligation Handler is specific to that given OH. Some will not require any properties beyond the standard set while other will need addition information.

See the *Obligation Handler* documentation for more information, in particular the *Gridmap Account Mapping Obligation Handler*.

## 5.2.2 Basic Configuration Options

### SERVICE section

Property	Description	Required?	Default Value
entityID	This is a unique identifier for the PEP. It must be a URI (URL or URN) and the same entity ID should be used for all PEP instances that make up a single logical PEP. If a URL is used it need not resolve to any specific webpage.	Y	None. Recommended value is a URL corresponding to the logical PEP service (e.g. <code>http://pep.example.org</code> ).
host-name	This is the hostname or IP address to which the service will bind.	Y	None
port	This is the port to which the service will bind.	N	8154
admin-Host	The hostname upon which the service will listen for admin commands.	N	127.0.0.1
admin-Port	This is the port upon which the service will listen for admin commands. This port is only available on the localhost (127.0.0.1).	N	8155
admin-Password	This is the password required to accompany admin commands. If unspecified than no password is required to run admin commands.	N	None
pips	This is a space separated list of the INI section names that configure policy information points (PIP) that the PEP will invoke upon the arrival of every request. PIPs are executed in the order listed by this property. See the <i>policy information point :ref:&lt;argus_peg_pip&gt;</i> documentation for more information.	N	None
obligation-Handlers	This is a space separated list of the INI section names that configure obligations handlers that the PEP will use to fulfill obligation requirements sent back by the PDP. See the <i>obligation handler</i> documentation for more information.	N	None



**PDP section**

Property	Description	Re- quired?	De- fault Value
pdps	A space separated list of PDP endpoint URLs. Endpoints will be tried in turn until one returns a successful response. This provides limited failover support. If more intelligent failover is necessary or load balancing is required, a dedicated load-balancer/failover appliance should be used.	Y	None
maximum-CachedResponses	The maximum number of responses from any PDP that will be cached. Setting this value to 0 (zero) will disable caching. The maximum amount of time a single response is cached is controlled by the <code>cachedResponseTTL</code> property described below.	N	500

**SECURITY section**

Property	Description	Required?	De- fault Value
servicePrivateKey	An absolute path to the file containing the unencrypted, PEM-encoded, private key used by this service.	Yes, if requests from the PEP client should be done over HTTPS.	None.
serviceCertificate	An absolute path to the file containing the unencrypted, PEM-encoded, certificate used by this service.	Yes, if requests from the PEP client should be done over HTTPS.	None.
trustInfoDir	An absolute path to the directory that contains standard X.509 trust information, such as the IGTF Trust Anchor Distribution.	Required when connecting to PDPs over HTTPS.	None
enableSSL	Enable HTTPS on the service port (SSL/TLS). The <code>serviceCertificate</code> , <code>servicePrivateKey</code> , and <code>trustInfoDir</code> properties must also be defined in order to use this setting.	N	false
require-ClientCertificateAuthentiation	The client must have a valid X.509 client certificate to authenticate to the PEP Server	N	true

**Example pepd.ini files**

The following example file contain the bare minimum required for a valid PEP configuration file.

```
[SERVICE]
entityId = https://argus.example.org/pep
hostname = argus.example.org

[PDP]
pdps = https://argus.example.org:8152/authz

[SECURITY]
```

The following example file contains the bare minimum required for a valid PEP configuration plus the configuration of a couple PIPs. Note how each element in the list `pips` list of the `SERVER` section matches the name section configuring the PIP. Also note that the `REQVALIDATOR_PIP` takes a few additional configuration parameters.

```

[SERVICE]
entityID = http://argus.example.org/pep
hostname = argus.example.org
pips = REQVALIDATOR_PIP

[PDP]
pdps = https://argus.example.org:8152/authz https://pdp2.example.org:8152/authz

[SECURITY]
servicePrivateKey = /etc/grid-security/hostkey.pem
serviceCertificate = /etc/grid-security/hostcert.pem
trustInfoDir = /etc/grid-security/certificates
enableSSL = true
requireClientCertAuthentication = true

[REQVALIDATOR_PIP]
parserClass = org.glite.authz.pip.provider.RequestValidatorPIPIniConfigurationParser
validateRequestSubjects = true
validateRequestResources = true
validateRequestAction = true
validateRequestEnvironment = false

```

### 5.2.3 Advanced Configuration Options

The following advanced options are available but are unlikely to ever be used by deployers. They are mostly for performing very fine-grained tuning of request/response handling parameters. Incorrectly configuring these can have a very negative impact on performance so deployers should not change these unless they are very sure they understand what the impact will be.

#### SERVICE section

Property	Description	Re-quired?	De-fault Value
maximumRe-quests	The maximum number of requests that will be processed simultaneously. Additional requests will be queued.	N	200
re-questQueueSize	The maximum number of requests that will be queued up when all the processing threads are busy. Incoming requests received when all processing threads are busy and the queue is full will receive an HTTP 503 error.	N	500
connec-tion-Timeout	This is the length of time, in seconds, the service will wait for the client to send information before it considers the request timed out.	N	30 seconds
receive-Buffer-Size	This is the size, in bytes, that will be allocated to the HTTP request buffer.	N	16384 (16kb)
send-Buffer-Size	This is the size, in bytes, that will be allocated to the HTTP response buffer.	N	16384 (16kb)

## PDP section

Property	Description	Re-quired	Default Value
maximum-Requests	The maximum number of simultaneous requests that will be made to the PDP. Additional requests will wait until a free request slot becomes available	No	200
cachedResponseTTL	The length of time, in seconds, for which a response will be cached	No	600 (10 minutes)
connectionTime-out	This is the length of time, in seconds, the PDP client will wait for the PDP to send information before it considers the request timed out	No	30
receive-BufferSize	This is the size, in bytes, that will be allocated to the PDP client send buffer	None	16384 (16 KB)
send-BufferSize	This is the size, in bytes, that will be allocated to the PDP client request buffer	None	16384 (16 KB)

## SECURITY section

Property	Description	Re-quired?	Default Value
trustInfoRefresh	The frequency, in minutes, that the trust material specified by <code>trustInfoDir</code> will be checked for updates.	N	60 (1 hour)

## 5.3 Argus PEP Server: Operation

### 5.3.1 EMI

For EMI Argus (v1.6) , see the [Argus Service Reference Card](#)

### 5.3.2 Starting & Stopping the Service

`$PEPD_HOME/sbin/pdpctl.sh start` starts the service

`$PEPD_HOME/sbin/pdpctl.sh stop` stops the services

`$PEPD_HOME/sbin/pdpctl.sh status` provides service status data

`$PEPD_HOME/sbin/pepdctl.sh clearResponseCache` causes the currently cached responses from the PDP can be flushed from memory

### 5.3.3 Service Information

#### Service Ports

- Default Service Port: 8154
- Default Admin Port: 8155

The PEP service only requires the standard service port to be open to those PEP clients which will communicate with the PEP. The PEP must also be able to make outbound connections to those PDPs to which is will make policy decision requests.

## Service Endpoints

This service contains the following endpoint URLs:

- `https://argus.example.org:8154/authz` - This endpoint is the recipient of authorization requests and is reachable on the standard service host and port.
- `http://127.0.0.1:8155/status` - This endpoint provides current status information on the PEP daemon and is reachable on the standard admin host and port.
- `http://127.0.0.1:8155/clearResponseCache` - This endpoint instructs the PEP daemon to flush its PDP response cache. It is reachable on the standard admin host and port.
- `http://127.0.0.1:8155/shutdown` - This endpoint instructs the PEP daemon to shutdown and is reachable on the standard admin host and port.

**Note:** Admin services may be password protected and thus not invokable without this password.

### 5.3.4 Logging and Logs

This service uses the `logback` logging library. Java developers are probably familiar with Apache Log4J, logback is written by the developer who initially wrote Log4J and contains a cleaner API and is much more performant. The configuration file for the logging system can be found in `$PEP_HOME/conf/logging.xml` and changes to this file are picked up every 5 minutes.

#### Enable Debug Logging

To enable debug logging follow: 1 Locate the line that contains `logger name="org.gluu.authz"` (line 10 in the default logging config) 1 On the following line, change `INFO` to `DEBUG`

In some cases it may be helpful to see the incoming and outgoing messages. To do this: 1 Locate the line that contains `org.gluu.authz.message.protocol` (line 15 in the default logging config) 1 On the following line, change `INFO` to `DEBUG`

**NOTE** always change your logging levels back to their original values once you are done debugging a problem. Keeping the system on the debug logging level could fill up your disk partition in a short time.

#### Service Logs

The service writes three different logs, located in `$PEPD_HOME/logs`:

- **process.log** - This log file contains the normal, human-oriented logging messages that the system generates while in operation. This can be thought of as the debug log (though that's really only true if configured to log debug messages).
- **access.log** - This file is an Apache-style access log showing information about incoming requests. This log is meant to be machine parsed. Each line contains the following ' | ' (pipe) delimited fields:
  - request time - time of the request, in the UTC timezone, from the Unix epoch (Jan 1, 1970, 00:00:00 UTC)
  - remote host - the hostname or IP address of the host that contacted the PDP
  - server host - the hostname or IP address on which the server received the request
  - server port - the port on which the server received the request
  - request path - the URL path that was requested

- **audit.log** - This file contains information useful for auditing the system (e.g. results of authorization requests). This log is meant to be machine parsed. Each line contains the following ‘|’ (pipe) delimited fields:
  - request time - time of the request, in the UTC timezone, from the Unix epoch (Jan 1, 1970, 00:00:00 UTC)
  - request ID - the ID of the authorization request
  - responder ID - the URL of the PDP that responded to the authorization request
  - response ID - the ID of the response message sent back from the PDP
  - policy decision - the authorization decision that was reached

## 5.4 Argus PEP Server: Troubleshooting

### 5.4.1 PEP Daemon Returns “Stale” Results

The PEPd keeps a short (10 minutes by default) response cache. So identical requests made within a short time period will always provide the same answer. If you’re testing this can be a pain. You can clear the cache using the `pepdctl clearResponseCache` command. You can also turn off the cache through the `maximumCachedResponses` documented in the [PEPd configuration](#). Just be sure to enable it again before you put the system under heavy load.

Note that the PDP also caches the policies it reads, so during testing you may also want to [configure](#) the PDP to more quickly pick up policies from the PAP via the `retentionInterval` option.

### 5.4.2 Testing a policy without submitting a job

When authoring new policies or troubleshooting an existing policies it can be helpful to mock up requests, instead of getting users to perform the request over and over as you diagnose the problem. The PEPd offers a C and Java command line tool. The [C tool](#) is useful for specifically testing cases where policies are based on the resource ID, action ID, subject ID, and FQAN attributes. The [Java tool](#) allows you to mock up any request.

Here is an example of using the C command line tool to test a job submission. It specifies the PEPd service, resource ID, action ID, user’s DN, and primary FQAN.

```
/opt/glite/bin/pepcli -v -x \
-p http://vesta.switch.ch:8154/authz \
-r http://authz-interop.org/xacml/resource/resource-type/wn \
-a http://authz-interop.org/xacml/action/action-type/execute-now \
-s "CN=Alessandro Usai,O=SWITCH,C=CH,DC=users,DC=switch,DC=grid,DC=quovadisglobal,DC=com" \
-f /dech \
```

## 5.5 Argus PEP Server Policy Information Points (PIP)

Policy Information Points (PIPs) are plugins to the authorization service that help populate and/or complete an authorization request. PIPs may rely on information already within the request or they may simply be able to self generate the data that they will add.

### 5.5.1 Request Validator PIP

**NOTE:** This PIP is new in Argus 1.3 (EMI).

The Request Validator PIP validates the incoming authorization request. It checks that the incoming authorization request contains at least one subject, one resource and one action, and that the attributes within them have at least one value, which is not null or an empty string.

## Configuration

1. Create a new INI section for you PIP (you may choose any valid INI section name. e.g. REQVALIDATOR\_PIP)
2. Into the PIP INI section add the `parserClass` property with the value `org.glite.authz.pep.pip.provider.RequestValidatorPIPIniConfigurationParser`
3. Configure how the authorization request must be validated

## PIP Configuration Properties

Property	Description	Re-quired?	Default Value
<code>validateRequestSubjects</code>	Require at least one subject with non-empty attribute values	N	true
<code>validateRequestRe-sources</code>	Require at least one resource with non-empty attribute values	N	true
<code>validateRequestAction</code>	Require one action with non-empty attribute values	N	true
<code>validateRequestEnviron-ment</code>	Require one environment with non-empty attribute values	N	false

## Example Configuration

The following example shows a PEP server configuration with the request validator PIP enabled. The PIP validates the incoming authorization request and checks that it contains at least one subject, at least one resource and one action, all with non-empty or null attributes values.

```
[SERVICE]
entityId = https://example.org/pep
hostname = example.org
pips = REQVALIDATOR_PIP

[PDP]
pdps = http://localhost:8152/authz

[SECURITY]
trustInfoDir = /etc/grid-security/certificates

[REQVALIDATOR_PIP]
parserClass = org.glite.authz.pep.pip.provider.RequestValidatorPIPIniConfigurationParser
validateRequestSubjects = true
validateRequestResources = true
validateRequestAction = true
validateRequestEnvironment = false
```

## 5.5.2 OpenSSL Subject Converter PIP

**NOTE:** This PIP is new in Argus 1.3 (EMI).

The OpenSSL Subject Converter PIP transforms on-the-fly an incoming request subject attribute *subject-id* and/or attribute *subject-issuer* value from the old, unsupported and wrong OpenSSL oneline format (e.g. “/C=CH/O=example.org/CN=John Doe”) into a correct RFC2253 format value (e.g. “CN=John Doe,O=example.org,C=CH”) with the correct datatype.

## Configuration

1. Create a new INI section for you PIP (you may choose any valid INI section name. e.g. OPENSSLSUBJECT\_PIP)
2. Into the PIP INI section add the `parserClass` property with the value `org.glite.authz.pep.pip.provider.OpenSSLSubjectPIPIniConfigurationParser`
3. Configure which subject attribute ID and datatype values must be transformed from the OpenSSL format into the RFC2253 format.

## PIP Configuration Properties

Property	Description	Re-quired?	Default Value
<code>opensslSubjectAttributeIDs</code>	The space separated list of subject attribute IDs containing an OpenSSL value to convert	No	<code>urn:oasis:names:tc:xacml:1.0:subject:subject-id</code> <code>http://glite.org/xacml/attribute/subject-issuer</code>
<code>opensslSubjectAttributeDatatypes</code>	The space separated list of subject attribute datatypes containing an OpenSSL value to convert	No	<code>http://www.w3.org/2001/XMLSchema#string</code>

## Example Configuration

The following example shows a PEP server configuration with the OpenSSL Subject Converter PIP enabled, and transforming both the subject attribute IDs `urn:oasis:names:tc:xacml:1.0:subject:subject-id` and `http://glite.org/xacml/attribute/subject-issuer`, with the datatype `http://www.w3.org/2001/XMLSchema#string` values from the OpenSSL oneline format into the RFC2253 format.

```
[SERVICE]
entityId = https://example.org/pep
hostname = example.org
pips = OPENSSLSUBJECT_PIP

[PDP]
pdps = http://localhost:8152/authz

[SECURITY]
trustInfoDir = /etc/grid-security/certificates

[OPENSSLSUBJECT_PIP]
parserClass = org.glite.authz.pep.pip.provider.OpenSSLSubjectPIPIniConfigurationParser
opensslSubjectAttributeIDs = http://glite.org/xacml/attribute/subject-issuer urn:oasis:names:tc:xacml:1.0:subject:subject-id
opensslSubjectAttributeDatatypes = http://www.w3.org/2001/XMLSchema#string
```

### 5.5.3 Grid Authorization Profile PIP

**NOTE: This is the default profile supported starting from Argus 1.2.**

This PIP allows the PEP client to send only the end-user certificate or proxy as lone Subject *Key-Info* attribute. It will then parse the certificate, extract all the information from the certificate required by the gLite Grid XACML Authorization Profiles, and populate the request with attributes found in the certificate/proxy.

This PIP implements the [XACML Grid Worker Node Authorization Profile \(v.1.0\)](#) and the [XACML Grid Computing Element Authorization Profile \(v.1.0\)](#) specifications.

#### Configuration

1. Create a new INI section for you PIP (you may choose any valid INI section name. e.g. GLITEXACMLPROFILE\_PIP)
2. Into the PIP INI section add the `parserClass` property with the value `org.glite.authz.pep.pip.provider.GLiteAuthorizationProfilePIPIniConfigurationParser`
3. To enable VOMS attribute certificate support add the `vomsInfoDir` property with a value corresponding to the absolute path of the VOMS `vomsdir`, traditionally `/etc/grid-security/vomsdir`.
4. If, in the SECURITY section, the `trustInfoDir` property is not already set, add it with a value of the absolute filesystem path of your IGTF trust bundle.
5. Configure which profile IDs are to be accepted.

#### PIP Configuration Properties

Property	Description	Re-quired?	Default Value
acceptedProfileIDs	The space separated list of accepted authorization profile IDs	No	None.
vomsInfoDir	The absolute path to the VOMS <code>vomsdir</code> directory.	Y	None.
vomsInfoRefresh	The refresh interval time in minutes of the <code>vomsInfoDir</code> directory.	No	60
requireCertificate	The request Subject attribute key-info MUST be present in the incoming request.	No	true
requireProxy	The request Subject attribute key-info MUST to be a proxy (PEM encoded proxy chain).	No	false

**NOTE: If the “acceptedProfileIDs” is not defined, then all profile IDs present in the request environment *\*profile-id\** attribute are accepted.**

#### Required Request Attributes

This PIP requires that the request environment contains a *profile-id* attribute with the profile identifier, and that the request subject contains the certificate, and its chain, that were used to authenticate to the service, in the *key-info* attribute:

- The Profile Identifier
  - **type:** Environment
  - **id:** <http://glite.org/xacml/attribute/profile-id>
  - **data type:** <http://www.w3.org/2001/XMLSchema#anyURI>



- **multiple values allowed:** no
- **description:** The profile ID implemented by the incoming request.
- The Certificate or Proxy Certificate (with chain)
  - **type:** Subject
  - **id:** urn:oasis:names:tc:xacml:1.0:subject:key-info
  - **data type:** <http://www.w3.org/2001/XMLSchema#string>
  - **multiple values allowed:** no
  - **description:** The PEM encoded certificate chain. No certificate order is assumed however all certificates must be version 3 certificates. Zero or one VOMS attribute certificate may also be included.

### Populated Effective Request Attributes

The PIP will process the request subject *key-info* attribute and populate the following attributes:

- The Subject Identifier
  - **type:** Subject
  - **id:** urn:oasis:names:tc:xacml:1.0:subject:subject-id
  - **data type:** urn:oasis:names:tc:xacml:1.0:data-type:x500Name
  - **multiple values allowed:** no
  - **description:** This is the Subject DN as given in the end-entity certificate. It is in RFC2253 format.
- The End-entity Certificate Issuer
  - **type:** Subject
  - **id:** <http://glite.org/xacml/attribute/subject-issuer>
  - **data type:** urn:oasis:names:tc:xacml:1.0:data-type:x500Name
  - **multiple values allowed:** yes
  - **description:** This is the Subject DN of the root CA and all subordinate CAs that signed within the end-entity certificate chain. It is in RFC2253 format.

If VOMS support is enabled and a VOMS certificate is included within a user's proxy certificate, the following attributes will be populated within the request:

- The VO Name
  - **type:** Subject
  - **id:** <http://glite.org/xacml/attribute/virtual-organization>
  - **data type:** <http://www.w3.org/2001/XMLSchema#string>
  - **multiple values allowed:** yes
  - **description:** The names of the VOs to which the user is a member. Currently there is only ever one value.
- The VOMS Primary FQAN
  - **type:** Subject
  - **id:** <http://glite.org/xacml/attribute/fqan/primary>
  - **data type:** <http://glite.org/xacml/datatype/fqan>

- **issuer:** DN of the attribute certificate issuer
- **multiple values allowed:** no
- **description:** The primary Fully Qualified Attribute Name (FQAN) for the subject
- The VOMS FQANs
  - **type:** Subject
  - **id:** <http://glite.org/xacml/attribute/fqan>
  - **data type:** <http://glite.org/xacml/datatype/fqan>
  - **multiple values allowed:** yes
  - **description:** All the Fully Qualified Attribute Name (FQAN)s for the subject

## Example Configuration

The following example shows a PEP Server configuration with the Grid authorization profile PIP enabled, and accepting both the <http://glite.org/xacml/profile/grid-ce/1.0> and the <http://glite.org/xacml/profile/grid-wn/1.0> XACML Grid authorization profiles.

```
[SERVICE]
entityId = https://example.org/pep
hostname = example.org
pips = GLITEXACMLPROFILE_PIP

[PDP]
pdps = http://localhost:8152/authz

[SECURITY]
trustInfoDir = /etc/grid-security/certificates

[GLITEXACMLPROFILE_PIP]
parserClass = org.glite.authz.pep.pip.provider.GLiteAuthorizationProfilePIPIniConfigurationParser
vomsInfoDir = /etc/grid-security/vomsdir
acceptedProfileIDs = http://glite.org/xacml/profile/grid-ce/1.0 http://glite.org/xacml/profile/grid-wn/1.0
```

## 5.5.4 Common XACML Authorization Profile PIP

**NOTE:** This profile is supported since Argus 1.6 (EMI-3).

This PIP allows the PEP client to send only the end-user certificate or proxy as lone Subject *Key-Info* attribute. It will then parse the certificate, extract all the information from the certificate required by the Common XACML Authorization Profile, and populate the request with attributes found in the certificate/proxy.

This PIP implements the [Common XACML Authorization Profile \(1.1.1\)](#) specifications.

### Configuration

1. Create a new INI section for you PIP (you may choose any valid INI section name. e.g. COMMONXACML-PROFILE\_PIP)
2. Into the PIP INI section add the `parserClass` property with the value `org.glite.authz.pep.pip.provider.CommonXACMLAuthorizationProfilePIPIniConfigurationPar`

3. To enable VOMS attribute certificate support add the `vomsInfoDir` property with a value corresponding to the absolute path of the VOMS `vomsdir`, traditionally `/etc/grid-security/vomsdir`.
4. If, in the `SECURITY` section, the `trustInfoDir` property is not already set, add it with a value of the absolute filesystem path of your IGTF trust bundle.
5. Configure which profile IDs are to be accepted, normally `http://dcsec.org/xacml/profile/common-authz/1.1`.

### PIP Configuration Properties

Property	Description	Required?	Default Value
<code>acceptedProfileIDs</code>	The space separated list of accepted authorization profile IDs	No	None.
<code>vomsInfoDir</code>	The absolute path to the VOMS <code>vomsdir</code> directory.	YES	None.
<code>vomsInfoRefresh</code>	The refresh interval time in minutes of the <code>vomsInfoDir</code> directory.	No	60
<code>requireCertificate</code>	The request Subject attribute key-info MUST be present in the incoming request.	No	false
<code>requireProxy</code>	The request Subject attribute key-info MUST to be a proxy (PEM encoded proxy chain).	No	false

**NOTE:** If the “`acceptedProfileIDs`” is not defined, then all profile IDs present in the request environment `*profile-id*` attribute are accepted.

### Required Request Attributes

This PIP requires that the request environment contains a *profile-id* attribute with the profile identifier, and that the request subject contains the certificate, and its chain, that were used to authenticate to the service, in the *key-info* attribute:

- **The Profile Identifier Attribute**
  - **type:** Environment
  - **id:** `http://dcsec.org/xacml/attribute/profile-id`
  - **data type:** `http://www.w3.org/2001/XMLSchema#anyURI`
  - **multiple values allowed:** no
  - **description:** The profile ID implemented by the incoming request, typically `http://dcsec.org/xacml/profile/common-authz/1.1`
- **The Subject Key-Info (certificate or proxy, with chain) Attribute**
  - **type:** Subject
  - **id:** `urn:oasis:names:tc:xacml:1.0:subject:key-info`
  - **data type:** `http://www.w3.org/2001/XMLSchema#base64Binary`
  - **multiple values allowed:** yes
  - **description:** The multiple values are the base64 encoded DER blocks of the certificate/proxy chain.

## Populated Effective Request Attributes

The PIP will process the request subject *key-info* attribute and populate the following attributes:

- The Subject Identifier Attribute
  - **type:** Subject
  - **id:** <urn:oasis:names:tc:xacml:1.0:subject:subject-id>
  - **data type:** <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
  - **multiple values allowed:** no
  - **description:** X.509 distinguished name of the end-entity certificate. The value is in RFC2253 format, e.g. “CN=John Doe,DC=example,DC=org”
- The Subject Issuer Attribute
  - **type:** Subject
  - **id:** <http://dc1-sec.org/xacml/attribute/subject-issuer>
  - **data type:** <urn:oasis:names:tc:xacml:1.0:data-type:x500Name>
  - **multiple values allowed:** yes
  - **description:** X.509 distinguished name of the authority(ies) which issued the end-entity certificate. The values are in RFC2253 format.

If VOMS support is enabled and a VOMS certificate is included within a user’s proxy certificate, the following attributes will be populated within the request:

- The Virtual Organization (VO) Attribute
  - **type:** Subject
  - **id:** <http://dc1-sec.org/xacml/attribute/virtual-organization>
  - **data type:** <http://www.w3.org/2001/XMLSchema#string>
  - **multiple values allowed:** yes
  - **description:** The names of the VOs to which the user is a member. Currently there is only ever one value.
- The Primary Group and Group Attributes
  - **type:** Subject
  - **id:** <http://dc1-sec.org/xacml/attribute/group/primary> and <http://dc1-sec.org/xacml/attribute/group>
  - **data type:** <http://www.w3.org/2001/XMLSchema#string>
  - **multiple values allowed:** no (primary group), yes (groups)
  - **description:** The primary group name, and the list of all group names
- The Primary Role and Role Attributes
  - **type:** Subject
  - **id:** <http://dc1-sec.org/xacml/attribute/role/primary> and <http://dc1-sec.org/xacml/attribute/role>
  - **data type:** <http://www.w3.org/2001/XMLSchema#string>
  - **issuer:** The group name to which this role belong.
  - **multiple values allowed:** no (primary role), yes (roles)
  - **description:** The primary role, and roles list assigned to the subject.

## Example Configuration

The following example shows a PEP Server configuration with the Common XACML authorization profile PIP enabled, and accepting the `http://dci-sec.org/xacml/profile/common-authz/1.1` EMI Common XACML Authorization profile.

```
[SERVICE]
entityId = https://argus.example.org/pep
hostname = argus.example.org

pips = COMMONXACMLPROFILE_PIP

[PDP]
pdps = https://argus.example.org:8152/authz

[SECURITY]
trustInfoDir = /etc/grid-security/certificates

[COMMONXACMLPROFILE_PIP]
parserClass = org.glite.authz.pep.pip.provider.CommonXACMLAuthorizationProfilePIPIniConfigurationParser
vomsInfoDir = /etc/grid-security/vomsdir
acceptedProfileIDs = http://dci-sec.org/xacml/profile/common-authz/1.1
```

## 5.5.5 Other Policy Information Points (PIP)

Here are other PIPs that you can configure for testing or debugging purpose

### Attribute White List PIP

This PIP can be used to filter out attributes that should not be accepted within a request.

#### Configuration

1. Create a new INI section for you PIP (you may choose any valid INI section name. e.g. `WHITELIST_PIP`)
2. Into the PIP INI section add the `parserClass` property with the value `org.glite.authz.pep.pip.provider.AttributeWhitelistPIPIniConfigurationParser`
3. Configure which request attributes are to be accepted

#### PIP Configuration Properties

Property	Description	Re-quired?	Default Value
<code>acceptedActionAttributes</code>	spaced delimited list attribute IDs that may appear in the request action	N	None.
<code>acceptedEnvrionmentAttributes</code>	spaced delimited list attribute IDs that may appear in the request environment	N	None.
<code>acceptedResourceAttributes</code>	spaced delimited list attribute IDs that may appear in the request resource	N	None.
<code>acceptedSubjectAttributes</code>	spaced delimited list attribute IDs that may appear in the request subject	N	None.

**NOTE:** if a property is not given then all attributes within the section (i.e. action, environment, resource, or subject) are accepted.

### Example Configuration

The following example shows a PEP Server configuration with the whitelist PIP enabled, accepting only the *key-info* attribute from the request subject, and filtering out all other subject attributes. All the other request attributes (action, resource and environment) are accepted as is.

```
[SERVICE]
entityId = https://example.org/pep
hostname = example.org
pips = WHITELIST_PIP

[PDP]
pdps = http://localhost:8152/authz

[SECURITY]
trustInfoDir = /etc/grid-security/certificates

[WHITELIST_PIP]
parserClass = org.glite.authz.pep.pip.provider.AttributeWhitelistPIPIniConfigurationParser
acceptedSubjectAttributes = urn:oasis:names:tc:xacml:1.0:subject:key-info
```

### Environment Time PIP

This PIP populates a few time-related attributes within the **environment** portion of the request.

**Note**, using this PIP effective disables the response caching in a PEP Server as will make every request different.

### Configuration

1. Create a new INI section for you PIP (you may choose any valid INI section name. e.g. TIME\_PIP)
2. To PIP INI section add the `parserClass` property with the value `org.glite.authz.pep.pip.provider.EnvironmentTimePIPIniConfigurationParser`
3. Add the name of the created PIP INI section to the list of PIPs in the SERVICE section

### Prerequisite Request Attributes

None.

### Populate Effective Request Attributes

This PIP will populate the following attributes within the environment portion of the request.

- The Current Time
  - **type:** environment
  - **id:** urn:oasis:names:tc:xacml:1.0:environment:current-time
  - **data type:** <http://www.w3.org/2001/XMLSchema#time>

- **issuer:** any
- **multiple values allowed:** no
- **description:** The time, in the UTC timezeon, the request was issued
- The Current Date
  - **type:** environment
  - **id:** urn:oasis:names:tc:xacml:1.0:environment:current-date
  - **data type:** <http://www.w3.org/2001/XMLSchema#date>
  - **issuer:** any
  - **multiple values allowed:** no
  - **description:** The date, in the UTC timezeon, the request was issued
- The Current Date and Time
  - **type:** environment
  - **id:** urn:oasis:names:tc:xacml:1.0:environment:current-dateTime
  - **data type:** <http://www.w3.org/2001/XMLSchema#dateTime>
  - **issuer:** any
  - **multiple values allowed:** no
  - **description:** The date and time, in the UTC timezone, the request was issued

## Example Configuration

The following example shows a PEP Server configuration with the Environment Time PIP enabled:

```
[SERVICE]
entityId = https://example.org/pep
hostname = example.org
pips = TIME_PIP

[PDP]
pdps = http://localhost:8152/authz

[TIME_PIP]
parserClass = org.glite.authz.pep.pip.provider.EnvironmentTimePIPIniConfigurationParser
```

## Static Attributes PIP

This PIP can populate the action, environment, resource, and subject of the request with a static set of attributes.

This PIP is very useful for testing as it allows for the creation of any arbitrary request.

## Configuration

1. Create a new INI section for you PIP (you may choose any valid INI section name)
2. To PIP INI section add the `parserClass` property with the value `org.glite.authz.pep.pip.provider.StaticPIPIniConfigurationParser`

3. Define the property `staticAttributesFile` with a fully qualified path to a file that will hold the definitions for the static attributes
4. If populating action attributes, define the property `actionAttributes` with a space delimited list of the INI sections, defined in the `staticAttributesFile` file, that represent the attributes that should be treated as action attributes.
5. If populating environment attributes, define the property `environmentAttributes` with a space delimited list of the INI sections, defined in the `staticAttributesFile` file, that represent the attributes that should be treated as environment attributes.
6. If populating resource attributes, define the property `resourceAttributes` with a space delimited list of the INI sections, defined in the `staticAttributesFile` file, that represent the attributes that should be treated as resource attributes.
7. If populating subject attributes, define the property `subjectAttributes` with a space delimited list of the INI sections, defined in the `staticAttributesFile` file, that represent the attributes that should be treated as subject attributes.
8. If the defined subject attributes should be added to each subject in the request, define the property `includeSubjectAttributesInAllSubjects` with a value of “true”
9. Optionally define the property `defaultAttributeIssuer` to a value that will be used as the attribute issuer if the attribute definition does not define an issuer.
10. Add the name of the created PIP INI section to the list of PIPs in the `SERVICE` section

To define your static attribute files, repeat the following steps for each static attribute you wish to define:

1. Create a new INI section for you PIP (you may choose any valid INI section name)
2. Define the property `id` with the value of the ID of the attribute
3. Optionally define the property `datatype` with the datatype of the attribute. If no datatype is define the default data type will be `http://www.w3.org/2001/XMLSchema#string`
4. Optionally define the property `issuer` with the ID of the issuer for the attribute.
5. Define the property `values` with a delimited string representing the values of the attribute. See next step for the delimiter.
6. Optionally define the property `valueDelimiter` with a delimiter string used to separate values in the previous property. If no delimiter is defined the default delimiter is ‘,’ (comma).

### Prerequisite

None.

### Populate Attributes

This PIP will populate those attributes defined in the `staticAttributesFile` file and referenced by either the `actionAttributes`, `environmentAttributes`, `resourceAttributes`, or `subjectAttributes` properties.

### Example Configuration

The following example shows a PEP Server configuration with the Static Attributes PIP enabled:



```
[SERVICE]
entityId = https://example.org/pep
hostname = example.org
pips = STATIC

[PDP]
pdps = http://localhost:8152/authz

[STATIC]
parserClass = org.glite.authz.pep.pip.provider.StaticPIPIniConfigurationParser
staticAttributesFile = /path/to/some/file.ini
actionAttributes = actionId
resourceAttributes = resourceId
subjectAttributes = subjectId
```

And here is a static attribute definition file, note that this file can include attributes which are not currently used as action, environment, resource, or subject attributes:

```
[actionId]
id = urn:oasis:names:tc:xacml:1.0:action:action-id
values = submit

[resourceId]
id = urn:oasis:names:tc:xacml:1.0:resource:resource-id
values = http://example.org/wn

[subjectId]
id = urn:oasis:names:tc:xacml:1.0:subject:subject-id
datatype = urn:oasis:names:tc:xacml:1.0:data-type:x500Name
values = CN=foo
```

## 5.6 Argus PEP Clients

### 5.6.1 pepcli

`pepcli` is the main ARGUS PEP client, based on the C API of the PEP client.

The `pepcli` command allows you to submit a XACML request to the PEP daemon and display the XACML response.

The command is very useful to check if a XACML policy applies (decision is Permit, Deny, Not Applicable or Indeterminate) for the given XACML request. Or to check if the XACML response correspond to the values you are expecting (`uidgid`, `secondary-gids`, ...) for the user mapping.

#### Installation

The `pepcli` command is by default installed with the `emi-argus` metapackage, see [Argus EMI Deployment](#). But you can also install it separately on your host for testing purpose. To install it with YUM:

```
yum install argus-pepcli
```

## Usage

### Usage:

```
pepcli --pepd <URL> --keyinfo <FILE> [options...]  
pepcli --pepd <URL> --subjectid <DN> [options...]
```

Submit a XACML Request to the PEPd and show the XACML Response.

### Options:

-p --pepd <URL>	Argus PEP server endpoint URL.
-k --keyinfo <FILE>	XACML Subject key-info: proxy or X509 file.
-s --subjectid <DN>	XACML Subject identifier: user DN (format RFC2253).
-f --fqan <FQAN>	XACML Subject primary FQAN and FQANs Add multiple --fqan options for secondary FQANs.
-r --resourceid <URI>	XACML Resource identifier.
-a --actionid <URI>	XACML Action identifier.
-t --timeout <SEC>	Connection timeout in second (default 30s).
-x --requestcontext	Show effective XACML Request context.
-v --verbose	Verbose.
-q --quiet	Turn off output.
-d --debug	Show debug information.
-h --help	This help.
-V --version	Display version and exit.

### TLS options:

--capath <DIR>	Directory containing the server PEM encoded CA certificates.
--cacert <FILE>	Server PEM encoded CA certificate filename.
--cert <FILE>	Client PEM encoded certificate filename.
--key <FILE>	Client PEM encoded private key filename.
--keypasswd <PASSWD>	Password of the client private key If the --keypasswd is omitted and the private key is encrypted, then you will be prompted for the password.

The MAN page is also available: `man pepcli`

## Return Code

The `pepcli` command return 0 when a valid XACML Response have been received, but neither content of the XACML response, nor the decision value are checked.

On error the `pepcli` command returns the following value:

- 1 on memory allocation error
- 2 invalid option
- 3 certchain file or content (missing certificate block) error
- 4 XACML request error
- 5 PEP-C client library error (see output for more detail)

## Examples

Submit a XACML Request to the PEP daemon endpoint URL `https://argus.example.org:8154/authz` (using SSL/TLS client authentication) for the resource identified by `my_resource_id` for the action `my_action_id`, using the Grid proxy as credentials `/tmp/x509up_u637`:

```
$ pepcli --pepd https://argus.example.org:8154/authz --capath /etc/grid-security/certificate --cert .
Decision: Permit
```

## 5.6.2 PEP Java Client Command Line

The PEP-J library contains a basic command line utility that is useful for sending test requests to the PEP daemon.

The command line client, `bin/testreq.sh`, takes a single parameter, the path (absolute or relative) path to a client configuration file. This client will create an empty request and then run any configured PIPs in order to populate it. The *Static Attribute PIP* is very useful for creating custom requests with which to test.

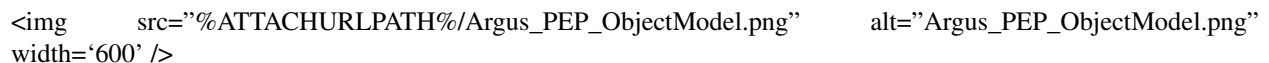
## 5.7 Argus PEP Client Library: C API

### 5.7.1 Documentation

The Doxygen documentation for the Argus PEP client library describes the C API and have an example.

API: <http://argus-authz.github.com/argus-pep-api-c/doc/modules.html>

### 5.7.2 PEP XACML Object Model

The image is a placeholder for a diagram titled "Argus\_PEP\_ObjectModel.png". It is intended to illustrate the XACML Object Model for the Argus PEP client library.

### 5.7.3 Basic Example

Basically, to use the Argus PEP client API, you will have to:

1. import the header with `#include "argus/pep.h"`
2. create and initialize the PEP client handle with `PEP * pep= pep_initialize()`
3. set the PEP Server URL with `pep_setopt(pep, PEP_OPTION_ENDPOINT_URL, "https://pepd.example.org")`
4. if the PEP Server URL is protected by HTTPS with client authentication (the default), you must also configure:
  - (a) the client certificate or proxy with `pep_setopt(pep, PEP_OPTION_ENDPOINT_CLIENT_CERT, "/tmp/x509...")`
  - (b) the client private key or proxy key with `pep_setopt(pep, PEP_OPTION_ENDPOINT_CLIENT_KEY, "/tmp/x509...")`
  - (c) the server CA trust anchors path with `pep_setopt(pep, PEP_OPTION_ENDPOINT_SERVER_CAPATH, "/etc/...")`
5. Optionally, you can register some [Policy Information Points \(PIP\)](#) and [Obligation Handlers \(OH\)](#) of your own with `pep_addpip(...)` and `pep_addobligationhandler(...)`
6. create a XACML Request and add the required Subject, Resource, Action and Environment to it with `xacml_request_create()`, `xacml_request_addsubject(request, subject)`, and so on. See the [PEP XACML Object Model](#) for the complete API.
7. submit the request and get the response: `pep_authorize(pep, &request, &response)`
8. process the response (if not already done by your obligation handlers)
9. release the PEP client handle with `pep_destroy(pep)`

### 5.7.4 Complex Example

A more detailed PEP client example is available [http://argus-authz.github.com/argus-pep-api-c/doc/pep\\_client\\_example\\_8c-example.html](http://argus-authz.github.com/argus-pep-api-c/doc/pep_client_example_8c-example.html)

### 5.7.5 Multi-threaded Programming

The Argus PEP client library is thread-friendly, but you are not allowed to share a PEP handle among multiple threads.

Each thread have to create its own PEP handle:

```
/* Each thread creates its own PEP handle */
PEP * pep= pep_initialize();
```

Within a thread you can reuse the PEP handle (multiple `pep_authorize(..)` calls).

If your threads are object (OO programming, ...), it is recommended you to create (`pep_initialize`) the PEP handle in the constructor, and release it (`pep_destroy`) in the destructor.

### 5.7.6 Processing Authorization Decision

The PEP client MUST abide by the authorization decision as described in here:

- If the decision is `Permit`, then the PEP client SHALL permit access. If **obligations** accompany the decision, then the PEP client SHALL permit access **only if it understands and it can and will enforce** those obligations.
- If the decision is `Deny`, then the PEP client SHALL deny access.
- If the decision is `NotApplicable`, meaning that no policy apply, then the PEP client SHALL deny access.
- If the decision is `Indeterminate`, then the PEP client SHALL deny access. The decision status message and status code should be used to produce an error message. Example:

```
...
xacml_result_t * result= xacml_response_getresult(response,i);
fprintf(stdout,"response.result[%d].decision= %s\n", i, decision_tostring(xacml_result_getdecision(result)));
fprintf(stdout,"response.result[%d].resourceid= %s\n", i, xacml_result_getresourceid(result));
if (xacml_result_getdecision(result) == XACML_DECISION_INDETERMINATE) {
    xacml_status_t * status= xacml_result_getstatus(result);
    fprintf(stdout,"response.result[%d].status.message= %s\n", i, xacml_status_getmessage(status));
    statuscode= xacml_status_getcode(status);
    fprintf(stdout,"response.result[%d].status.code.value= %s\n", i, xacml_statuscode_getvalue(status));
}
...
```

## 5.8 Argus PEP Client: Java Programming Interface

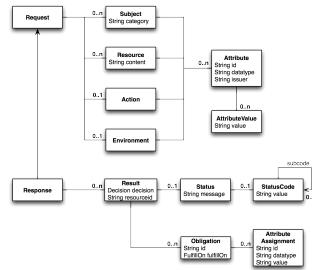
The Argus PEP Java client library is used to communicate with the Argus PEP daemon. It authorizes request and receives response back from Argus.

### 5.8.1 Javadoc

Javadoc for the PEP Java client API: <http://argus-authz.github.com/argus-pep-api-java/javadoc/2.X/index.html>

## 5.8.2 PEP XACML Object Model

The PEP client XACML object model implemented in the package `org.glite.authz.common.model` follow this schema:



## 5.8.3 Basic Example

This is a very simplified example, omitting all the error handling, on how to create a PEP client, a request, and then authorize the request and process the response.

1. Create a PEP client configuration and initialize it:

```
PEPClientConfiguration config= new PEPClientConfiguration();
config.addPEPDaemonEndpoint("https://argus.example.org:8154/authz");
// trust and key material for the HTTPS connection with client authentication
config.setTrustMaterial("/etc/grid-security/certificates");
config.setKeyMaterial("/etc/grid-security/hostcert.pem", "/etc/grid-security/hostkey.pem", "keystore");
```

2. Create the PEP client based on the config:

```
PEPClient pep= new PEPClient(config);
```

At this point you have a multi-threaded PEP client that can be **reuse** to submit many authorization requests to the PEP server.

3. Create an authorization request for a user proxy certificate, based on a profile:

```
// read the user proxy
PEMFileReader reader= new PEMFileReader();
X509Certificate[] userproxy= reader.readCertificates("/tmp/x509up_u959");
// create the request for a given profile
AuthorizationProfile profile= GridWNAuthorizationProfile.getInstance();
Request request= profile.createRequest(userproxy,
                                     "http://example.org/wn/cluster1",
                                     GridWNAuthorizationProfile.ACTION_EXECUTE);
```

4. Authorize the request with the Argus PEP daemon:

```
Response response= pep.authorize(request);
```

5. Extract the user mapping information from the response:

```
// will throw an exception if the authorization response is not *Permit*, or if the obligation is not
Obligation posixMappingObligation= profile.getObligationPosixMapping(response);
String userId= profile.getAttributeAssignmentUserId(posixMappingObligation);
String groupId= profile.getAttributeAssignmentPrimaryGroupId(posixMappingObligation);
List<String> groupIds= profile.getAttributeAssignmentGroupIds(posixMappingObligation);
```

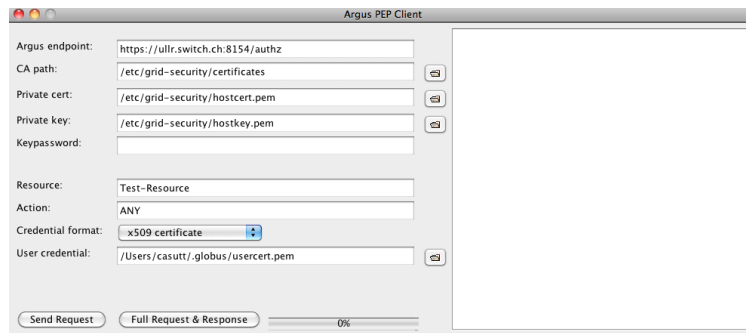
## 5.8.4 Processing Authorization Decision

The PEP client MUST abide by the authorization decision as described in here:

- If the decision is `Permit`, then the PEP client SHALL permit access. If **obligations** accompany the decision, then the PEP client SHALL permit access **only if it understands and it can and will enforce** those obligations.
- If the decision is `Deny`, then the PEP client SHALL deny access.
- If the decision is `NotApplicable`, meaning that no policy apply, then the PEP client SHALL deny access.
- If the decision is `Indeterminate`, then the PEP client SHALL deny access. The decision status message and status code should be used to produce an error message.

## 5.8.5 GUI

There is also a Java-based GUI available for sending requests to a PEPd. Just click on the following image which will download the application to your desktop and start it. Once you've downloaded it you can restart it by double-clicking the `Argus-PEP-Client.jnlp` file.



## 5.9 Argus GSI PEP Callout

### 5.9.1 Module Description

The Globus Toolkit version 3.2 and later have the ability to customize the authorization and gridmap lookup (currently available in the GridFTP and Gatekeeper servers). The [Globus Authorization Callouts](#) framework allows to plug in authorization and mapping modules.

The GSI PEP Callout module implements the functionality to authorize and map the user by calling out to the [Argus PEP Server](#).

### Authorization and Mapping

Based on the Grid credentials, typically a proxy certificate, the GSI PEP Callout module will send an authorization request to the [Argus PEP Server](#), and then parse the authorization response decision to authorize the user and the obligations to map him to a local account.

## XACML Profile

The GSI PEP Callout module implements the [XACML Grid Worker Node Authorization Profile 1.0](#), and by default uses the identifiers described in the profile.

## XACML Request

The GSI PEP Callout module sends a request to the PEP Daemon with the following elements:

- XACML subject with the attribute element:
  - AttributeId: `urn:oasis:names:tc:xacml:1.0:subject:key-info`
  - Value: The PEM encoded Grid credentials provided by the calling service
- XACML resource with the attribute element:
  - AttributeId: `urn:oasis:names:tc:xacml:1.0:resource:resource-id`
  - Value: The value of the GSI PEP Callout *configuration directive* `xacml_resourceid`.
- XACML action with the attribute element:
  - AttributeId: `urn:oasis:names:tc:xacml:1.0:action:action-id`
  - Value: The service being requested by the client (e.g. `file` for GridFTP) or the name of the service passed to the gatekeeper.
- XACML environment with the attribute element:
  - AttributeId: `http://glite.org/xacml/attribute/profile-id`
  - Value: `http://glite.org/xacml/profile/grid-wn/1.0` (Default)

## XACML Response

The PEP Daemon sends back a response to the GSI PEP Callout module. The following response elements are parsed to authorize and map the user:

- XACML decision element: Contains the **authorization decision** `Permit`, `Deny`, `Indeterminate` or `NotApplicable`
- XACML obligation, `ObligationId`: `http://glite.org/xacml/obligation/local-environment-map/posix`, with the attribute assignment element:
  - AttributeId: `http://glite.org/xacml/attribute/user-id`
  - Value: Contains the **local identity mapping** of the user.

The local identity mapping will only succeed if the authorization decision is `Permit`.

## 5.9.2 Configuration

### Manual Configuration

To configure the GSI PEP Callout module, you first have to configure the Globus Authorization Callouts framework to use the GSI PEP Callout library to do the authorization and mapping. Then the GSI PEP Callout module must be configured.

### Globus Authorization Callouts Configuration

Configuration file and configuration directives for the [Globus Authorization Callouts](#) to enable the GSI PEP Callout module.

**Configuration File** The Globus Authorization Callouts framework uses the following locations (in order) for the callout configurations file:

- `$GSI_AUTHZ_CONF` (Environment variable)
- `/etc/grid-security/gsi-authz.conf`
- `$GLOBUS_LOCATION/etc/gsi-authz.conf`
- `$HOME/.gsi-authz.conf`

**EMI-1 Configuration Directives** Content of the Globus Authorization Callouts configuration file to enable the GSI Argus PEP Callout function `argus_peg_callout` for EMI:

```
# Globus authorization and mapping callout to the ARGUS GSI PEP Callout module
# format: globus_mapping <library_path> <function_name>
globus_mapping /usr/lib64/libgsi_peg_callout.so argus_peg_callout
```

For EMI the Argus PEP GSI callout library is installed in the `/usr/lib64` directory.

### GSI PEP Callout Configuration

Configuration file and configuration directives for the GSI PEP Callout module.

**Configuration File** The GSI PEP Callout module uses the following locations (in order) for the configurations file:

- `$GSI_PEP_CALLOUT_CONF` (Environment variable)
- `/etc/grid-security/gsi-peg-callout.conf`

**Configuration Directives** The configuration directives for the GSI PEP Callout are single `name value` lines. Lines with comments `#` are allowed.



Directive	Description	Mandatory?	Default Value	Example	Since
pep_url	The endpoint URL of the PEP daemon.	Yes		pep_url https://pepd.example.org:8154/authz	1.0
xacml_resourceid	XACML request resource-id value	Yes		xacml_resourceid x-urn:example.org:resource:ce:gridftp	1.0
xacml_actionid	XACML request action-id value. Define this parameter to overwrite the <i>service</i> name passed to the module by the application	No		xacml_actionid http://glite.org/xacml/action/authz	1.0
xacml_profileid	XACML request profile-id value. Define this parameter to overwrite the default profile id	No	http://glite.org/xacml/profileid	http://glite.org/xacml/profile/grid-ce/1.0	1.2
pep_timeout	Connection timeout in seconds	No	30	pep_timeout 60	1.0
pep_ssl_validation	Enable SSL validation of the PEP daemon endpoint URL (HTTPS)	No	true	pep_ssl_validation false	1.0
pep_ssl_ca_dir	CA directory path for the HTTPS validation of the PEP daemon endpoint URL	No	/etc/grid-security/certificates	pep_ssl_ca_dir /etc/grid-security/certificates	1.0
pep_ssl_server_cert	Certificate file for the HTTPS validation of the PEP daemon endpoint URL	No		pep_ssl_server_cert /etc/grid-security/pepdcert.pem	1.0
pep_ssl_client_cert	Client certificate file for the TLS client authentication on the PEP daemon endpoint URL	No	/etc/grid-security/hostclient.pem	pep_ssl_client_cert /etc/ssl/mycert.pem	1.0
pep_ssl_client_key	Client private key file for the TLS client authentication on the PEP daemon endpoint URL	No	/etc/grid-security/hostkey.pem	pep_ssl_client_key /etc/ssl/mykey.pem	1.0
pep_ssl_server_key_passwd	Client private key password	Only if pep_ssl_client_key is encrypted		pep_ssl_server_key_passwd mykeypassword	1.0

**Configuration Example** Example of a valid configuration file for the GSI PEP Callout module:

```
#
# GSI PEP Callout configuration example
#
pep_url https://chaos.switch.ch:8154/authz
xacml_resourceid http://ce.example.org/cream/gridftp
```

## YAIM Configuration

The yaim-core (>= 4.0.12) the function config\_lcas\_lcmaps\_gt4 is now able to configure the Argus GSI PEP callout module.

In your `site-info.def` set the following variables:

```
USE_ARGUS=yes
ARGUS_PEPD_ENDPOINTS="<Argus_URL> ..."
CREAM_PEPD_RESOURCEID=<CreamCE_XACML_resource_id>
```

where `Argus_URL` is the Argus PEP daemon endpoint URL. e.g. `ARGUS_PEPD_ENDPOINTS=https://argus.example.org:`

where `CreamCE_XACML_resource_id` is the XACML resource identifier for this cream CE. e.g. `CREAM_PEPD_RESOURCEID=http://glite.org/xacml/resource/cream-ce`

### 5.9.3 Troubleshooting

#### Syslog

By default the GSI PEP Callout module logs info and error via syslog.

The syslog facility used is `local5` and the identifier is `gsi_peg_callout`. These log messages are typically in `/var/log/messages`

#### Enabling Debug Information

You can enable the debugging mode of the GSI PEP Callout module to troubleshoot your problem.

#### Environment Variables

You can set the following environment variables to enable debug mode:

- `GSI_PEP_CALLOUT_DEBUG_LEVEL` Set the debug level from 0 (none) to 9 (lot of info). Default is 0.
- `GSI_PEP_CALLOUT_DEBUG_FILE` Set the file to log the debugging information. Default is `stderr`.

#### Example

This example shows how to start the GridFTP server in debug mode. The configuration files `gsi-authz.conf` and `gsi-peg-callout.conf` must be correctly configured as *previously described*.

```
export GLOBUS_CALLOUT_DEBUG_LEVEL=5
# set the gsi-authz config to use (default /etc/grid-security/gsi-authz.conf)
export GSI_AUTHZ_CONF=/etc/grid-security/gsi-authz.conf
# set the gsi-peg-callout config to use (default /etc/grid-security/gsi-peg-callout.conf)
export GSI_PEP_CALLOUT_CONF=/etc/grid-security/gsi-peg-callout.conf
export GSI_PEP_CALLOUT_DEBUG_LEVEL=5

globus-gridftp-server -d 255 -p 9999 -debug
```

The GridFTP server is now running and listening on port 9999. Use the `uberftp` client or `globus-url-copy` to connect to the server with your Grid credentials and obtain debugging information from the server:

- `uberftp -P 9999 HOSTNAME`
- `globus-url-copy file:///etc/passwd gsiftp://HOSTNAME:9999/tmp/e33`

## 5.10 Argus PEP Server Obligation Handlers

Obligation handlers are plugins used by the authorization service in order to adjust the environment, under which an action will run, to a state that meets a particular set of obligations. Obligation are things like “write output to directory X” or “perform all work as user 1”.

### 5.10.1 Grid Map POSIX Account Mapping Obligation Handler

---

**Note:** The Grid Map Account Mapping Obligation Handler only works with the *Grid Authorization Profile PIP*, or with clients implementing the XACML Grid Worker Node Authorization Profile (v.1.0) or the XACML Grid Computing Element Authorization Profile (v.1.0) specifications.

---

This obligation handler maps a subject ID, given as a DN, and set of FQANs, for example those provided by the *Grid Authorization Profile PIP*, in to a POSIX account (login name, primary group name and secondary group names).

This mapping is controlled by two *grid map files*, one that provides the mapping of the subject to an account indicator (login name or pool account indicator) and one that maps the subject to a set of group names (one primary and any number of secondary).

## Configuration

1. Create the account and group *mapping files* appropriate for your environment
2. Create a new INI section for your handler (you may choose any valid INI section name)
3. To handler INI section add the `parserClass` property with the value `org.glite.authz.pep.obligation.dfpmap.DFPMObligationHandlerConfigurationParser`
4. Define at least the required ( `accountMapFile`, `groupMapFile`, `gridMapDir`) and any optional configuration properties for this handler

## OH Configuration Properties

Property	Description	Re- quired?	Default Value	Since
accountMapFile	The absolute path to the <i>map file</i> used to map a subject to a POSIX login name.	Y	None.	Argus 1.0
groupMapFile	The absolute path to the <i>map file</i> used to map a subject to a set of POSIX groups.	Y	None.	Argus 1.0
gridMapDir	The absolute path to the grid map directory.	Y	None.	Argus 1.0
handledObligationId	The identifier of the obligation handled by the handler. The obligation handler is triggered only if the obligation ID match this value.	N	<a href="http://glite.org/xacml/obligation/local-environment-map">http://glite.org/xacml/obligation/local-environment-map</a>	Argus 1.5
preferDNForLoginName	Indicates whether to prefer a DN based mapping for the login name mapping over a primary FQAN login name mapping.	N	true	Argus 1.1
preferDNForPrimaryGroup	Indicates whether to prefer a DN based mapping for the primary group name mapping over a primary FQAN group name mapping	N	true	Argus 1.2
noPrimaryGroupMapping	Indicates that the failure to find a primary group mapping in the group map file cause the obligation handler to fail..	N	false	Argus 1.2
refreshPeriod	The period, in minutes, between when the map files are checked, and if they have been changed, reread.	N	15	Argus 1.0
requireSubject	The obligation handler will only be applied if the request subject contains a <i>key-info</i> attribute (PEM encoded certificate)	N	true	Argus 1.3.0
useSecondaryGroups	The obligation handler will create lease file names containing the secondary groups of the user	N	true	Argus 1.3.1

**NOTE:** the default “preferDNForLoginName“ property value was *\*false\** for Argus 1.0 and Argus 1.1. From Argus 1.2 the default is *\*true\**.

## Required Response Obligation Trigger

This obligation handler is triggered if the PDP response contains the obligation <http://glite.org/xacml/obligation/local-environment-map>, or the value defined by the *handledObligationId* parameter.

## Required Request Attributes

This obligation handler **requires** the following request attributes in order to correctly map the user. The attributes can be provided by the *Grid Authorization Profile PIP*, or directly by the clients implementing the *XACML Grid Worker Node Authorization Profile (v.1.0)* or the *XACML Grid Computing Element Authorization Profile (v.1.0)* specifications.

- The Subject Identifier

- **type:** Subject
- **id:** urn:oasis:names:tc:xacml:1.0:subject:subject-id
- **data type:** urn:oasis:names:tc:xacml:1.0:data-type:x500Name
- **multiple values allowed:** no
- **description:** This is the Subject DN as given in the end-entity certificate. It is in RFC2253 format.
- The VOMS Primary FQAN
  - **type:** Subject
  - **id:** <http://glite.org/xacml/attribute/fqan/primary>
  - **data type:** <http://glite.org/xacml/datatype/fqan>
  - **issuer:** DN of the attribute certificate issuer
  - **multiple values allowed:** no
  - **description:** The primary Fully Qualified Attribute Name (FQAN) for the subject
- The VOMS FQANs
  - **type:** Subject
  - **id:** <http://glite.org/xacml/attribute/fqan>
  - **data type:** <http://glite.org/xacml/datatype/fqan>
  - **multiple values allowed:** yes
  - **description:** All the Fully Qualified Attribute Name (FQAN)s for the subject

## Response Obligation Results

This result of this obligation handler is the **replacement** of the generic `http://glite.org/xacml/obligation/local-environment-map` with the more specific obligation `http://glite.org/xacml/obligation/local-environment-map/posix`.

This later obligation carries the account and group names in the following attribute assignments:

- one `http://glite.org/attribute/xacml/user-id` account login name
- zero or one `http://glite.org/attribute/xacml/group-id/primary` primary group name
- zero or more `http://glite.org/xacml/attribute/group-id` secondary group names

## Example Configuration

This is an example PEPd configuration file with one gridmap POSIX account mapping obligation handler defined:

```
[SERVICE]
entityId = http://argus.example.org/pep
hostname = argus.example.org
obligationHandlers = ACCOUNT_MAPPING_OH

[PDP]
pdps = https://argus.example.org:8152/authz

[ACCOUNT_MAPPING_OH]
parserClass = org.glite.authz.pep.obligation.dfpmmap.DFPMObligationHandlerConfigurationParser
```

```
accountMapFile = /etc/grid-security/grid-mapfile
groupMapFile = /etc/grid-security/group-mapfile
gridMapDir = /etc/grid-security/gridmapdir
```

## Account and Group Mapping

This Obligation Handler uses the following logic to determine the mapping of the subject to a POSIX account.

### Preconditions

- The input to this process is the subject DN of the end-entity certificate of the user and optionally a primary FQAN and a list of secondary FQANs.
- When dealing with the account map file (the gridmap file) and the group map file, entries are evaluated in the order listed in the file. Once a match is found processing stops.
- The grid map directory is populated with information for all configured pool accounts on the system. A pool account is considered “configured” if there is a zero-byte file, whose name is the pool account name, in the grid map directory. The grid map directory must also be read/writable by the user running the authorization service.

### Mapping Steps

1. If a primary FQAN is given it is checked against the mappings listed in the account map file. If the primary FQAN matches a key in the map file then the associated value provides the account indicator.
2. If no account indicator was determined by means of the primary FQAN the subject DN is checked against the mappings listed in the account map file. If the DN matches a key in the map file then the associated value provides the account indicator. If no match is found processing stops and no map is available.
3. If the account indicator starts with a period (‘.’), its value, without the period, is considered to be a pool account name prefix. If the account indicator does not start with a period it is a POSIX account name. If no account indicator was determined the mapping process fails.
4. If a primary FQAN is given then it is evaluated against entries in a group map file. The first entry that matches determines the primary group name. If no match is found, the mapping process fails.
5. If one or more secondary FQANs are given then each one is matched against the group map file and each match determines a secondary group name. If no matches occur then there are no secondary group names associated with the account.
6. If the account indicator is a POSIX account name, and zero or one primary group and zero or more secondary group names were determined then the mapping is completed. The user is mapped to that account.
7. If the account indicator is a pool account name prefix a lookup in the grid map directory occurs. The file looked for is generated according to the template `encoded_dn{:primary_group_name{:secondary_group_name}*}?`  with the secondary group names listed in ascending alphabetical order.
8. If the file exists and has a link count of 2 then the hard link is followed to a file whose name is used as the POSIX account name. The last modified time of the two files is updated to the current time. If the link count is not 2 or the POSIX account name does not start with the account name prefix, the mapping process fails.
9. If the file does not exist, a list of files, within the grid map directory, matching the pool account name prefix followed exclusively by one or more numeric digits, is retrieved. An example regular expression representation of this would be `prod[0-9]+`

10. This list of files is searched for a file whose link count is 1, this is a candidate pool account file. A hard link to the pool account file, whose name corresponds to the filled in template described above is then created. If no such pool account file is found the mapping process fails.
11. The pool account file link count is rechecked, if it is more than 2 (indicating another request mapped to that same file at the same time) the created link is removed and the mapping process reverts to the previous step.
12. If the link count is 2 the mapping is complete. The user is mapped to the pool account corresponding to the given file.

**Note:** In the case where no FQANs are available, this obligation handler only returns a login name. No group information is returned.

### DN Encoding Rules

1. Leading zeros are removed from attribute types that are encoded as dotted decimal OIDs
2. DirectoryString attribute values of type PrintableString and UTF8String are not output in hexadecimal format
3. DirectoryString attribute values of types other than PrintableString and UTF8String are output in hexadecimal format
4. Leading and trailing white space characters are removed from non-hexadecimal attribute values (unless the value consists entirely of white space characters)
5. Internal substrings of one or more white space characters are converted to a single space in non-hexadecimal attribute values
6. Relative Distinguished Names containing more than one Attribute Value Assertion (AVA) are output in the following order: an alphabetical ordering of AVAs containing standard keywords, followed by a numeric ordering of AVAs containing OID keywords.
7. The only characters in attribute values that are escaped are those which section 2.4 of RFC 2253 states must be escaped (they are escaped using a preceding backslash character)
8. The entire name is converted to upper case using US localization
9. The entire name is converted to lower case using US localization
10. The name is finally normalized using normalization form KD, as described in the Unicode Standard and UAX #15
11. The DN is converted to the non-standard, openssl one line format
12. The string is then URL-encoded

---

### Note:

- A reorder of entries in the group map file that results in a change in the primary group will cause an incoming user to be mapped to a new account as this information is part of the link created in the grid map directory.
  - A reordering of entries in the group map file that does **not** result in a change to the primary group but provides the same secondary groups in a different order does **not** result in a new mapping for a returning user since the link created in the grid map directory orders the secondary group names in ascending alphabetical order.
-





---

## Argus Monitoring

---

### 6.1 Nagios Probes for Argus (UMD)

#### 6.1.1 Installation

##### Prerequisites

- Python 2.4 or newer (not Python 3000)
- UMD-3 or later (or EMI-3) repository installed
- Host certificate & IGTF-bundle
- Nagios server

##### Where to install

- This set of probes needs to be installed on the Nagios server only. On the client side (where the Argus services actually run), no action is required.

##### Install instructions

There is available a rpm called “nagios-plugins-argus”, ([nagios-plugins-argus - v. 1.1.0 \(EMI-3\)](#)). It can be installed as follows:

```
yum install nagios-plugins-argus
```

This installs the plugins into the `/usr/libexec/grid-monitoring/probes/` directory. It is up to the user to bind them into the local Nagios environment.

**NOTE:** If you use iptables you need to open the ports on client side (where the Argus services are installed):

for IPv4 do:

```
iptables -I INPUT 1 -p tcp --dport 8154 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 8152 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 8150 -j ACCEPT
/etc/init.d/iptables save
/etc/init.d/iptables restart
```

for IPv6 do:

```

iptables -I INPUT 1 -p tcp --dport 8154 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 8152 -j ACCEPT
iptables -I INPUT 1 -p tcp --dport 8150 -j ACCEPT
/etc/init.d/iptables save
/etc/init.d/iptables restart

```

## 6.1.2 Overview of probes

There are 9 different probes available, three for each service:

Service	Probe-Name
PAP	nagios-plugins-argus.PAP.memory
^	nagios-plugins-argus.PAP.policies
^	nagios-plugins-argus.PAP.status
PDP	nagios-plugins-argus.PDP.memory
^	nagios-plugins-argus.PDP.status
^	nagios-plugins-argus.PDP.traffic
PEPD	nagios-plugins-argus.PEP.memory
^	nagios-plugins-argus.PEP.status
^	nagios-plugins-argus.PEP.traffic

### PAP probes

#### nagios-plugins-argus.PAP.status

checks if the service is available. This is done by a call to <https://%60hostname:8150/pap/status>, if there is a valid response the service is considered up and running.

	Mandatory	CLI option	Description	Default
Options	YES	– hostname=HOSTNAME	The hostname of the service	
		–cert CERT	The SSL client certificate	/etc/grid-security/hostcert
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostkey
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certificates
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. <a href="https://hostname:port/status">https://hostname:port/status</a>	
		–version	show program's version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

#### nagios-plugins-argus.PAP.memory

the response of the call executed by the status probe is parsed. The value for the current memory consumption is read out and displayed in the Status information field of the Nagios control panel and written into the Performance Data tab of Nagios. Hence it can be displayed as a graph over time by Nagios add-ons, e.g. pnp.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–warning MEM_WARN	Memory usage warning threshold in MB	224.0
		–critical MEM_CRIT	Memory usage critical threshold in MB	256.0
		–cert CERT	The SSL client certificate	/etc/grid-security/hostce
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostke
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certif
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. <code>https://hostname:port/status</code>	
		–version	show program's version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

#### nagios-plugins-argus.PAP.policies

the response of the call executed by the status probe is parsed. The value for the current number of policies in the monitored PAP is read out and displayed in the Status information field of the Nagios control panel and written into the Performance Data tab of Nagios. Hence it can be displayed as a graph over time by Nagios add-ons, e.g. pnp.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–cert CERT	The SSL client certificate	/etc/grid-security/hostce
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostke
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certif
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. <code>https://hostname:port/status</code>	
		–version	show program's version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

#### PDP probes

##### nagios-plugins-argus.PDP.status

checks if the service is available. This is done by a call to `https://%60hostname':8152/status`, if there is a valid response the service is considered up and running.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–cert CERT	The SSL client certificate	/etc/grid-security/hostce
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostke
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certif
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. https://hostname:port/status	
		–version	show program’s version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

#### nagios-plugins-argus.PDP.memory

the response of the call executed by the status probe is parsed. The value for the current memory consumption is read out and displayed in the Status information field of the Nagios control panel and written into the Performance Data tab of Nagios. Hence it can be displayed as a graph over time by Nagios add-ons, e.g. pnp.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–warning MEM_WARN	Memory usage warning threshold in MB	224.0
		–critical MEM_CRIT	Memory usage critical threshold in MB	256.0
		–cert CERT	The SSL client certificate	/etc/grid-security/hostce
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostke
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certif
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. https://hostname:port/status	
		–version	show program’s version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

#### nagios-plugins-argus.PDP.traffic

the response of the call executed by the status probe is parsed. The value for the current number of requests executed and the number how many of them have been successful is read out and compared to the same numbers of the former test executed. Out of this the requests per second, the completed requests per second and the erroneous requests per second are computed and written into the Performance Data tab of Nagios. Hence it can be displayed as a graph over

time by Nagios add-ons, e.g. pnp. Additionally the number of requests since the last test is written into the Status information field of the Nagios control panel.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–cert CERT	The SSL client certificate	/etc/grid-security/hostcert.pem
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostkey.pem
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certificates
		– tempdir=TEMP_DIR	Storage path for the needed temporary file	../../../../var/lib/grid-monitorin
		– tempfile=TEMP_FILE	Name for the needed temporary file	hostname.nagios-plugins-argus.PDP.traffic.pickle
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. https://hostname:port/status	
		–version	show program's version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

## PEP Server probes

### nagios-plugins-argus.PEP.status

checks if the service is available. This is done by a call to <https://%60hostname':8154/status>, if there is a valid response the service is considered up and running.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–warning MEM_WARN	Memory usage warning threshold in MB	224.0
		–critical MEM_CRIT	Memory usage critical threshold in MB	256.0
		–cert CERT	The SSL client certificate	/etc/grid-security/hostce
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostke
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certif
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. https://hostname:port/status	
		–version	show program's version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

#### nagios-plugins-argus.PEP.memory

the response of the call executed by the status probe is parsed. The value for the current memory consumption is read out and displayed in the Status information field of the Nagios control panel and written into the Performance Data tab of Nagios. Hence it can be displayed as a graph over time by Nagios add-ons, e.g. pnp.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–warning MEM_WARN	Memory usage warning threshold in MB	224.0
		–critical MEM_CRIT	Memory usage critical threshold in MB	256.0
		–cert CERT	The SSL client certificate	/etc/grid-security/hostce
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostke
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certif
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. https://hostname:port/status	
		–version	show program's version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False

**nagios-plugins-argus.PEP.traffic**

the response of the call executed by the status probe is parsed. The value for the current number of requests executed and the number how many of them have been successful is read out and compared to the same numbers of the former test executed. Out of this the requests per second, the completed requests per second and the erroneous requests per second are computed and written into the Performance Data tab of Nagios. Hence it can be displayed as a graph over time by Nagios add-ons, e.g. pnp. Additionally the number of requests since the last test is written into the Status information field of the Nagios control panel.

Options	Mandatory	CLI option	Description	Default
	YES	– hostname=HOSTNAME	The hostname of the service	
		–cert CERT	The SSL client certificate	/etc/grid-security/hostcert.pem
		–key KEY	The private key (the key must be unencrypted)	/etc/grid-security/hostkey.pem
		–capath CAPATH	The directory where trust anchors are stored on the system	/etc/grid-security/certificates
		– tempdir=TEMP_DIR	Storage path for the needed temporary file	../../../../var/lib/grid-monitorin
		– tempfile=TEMP_FILE	Name for the needed temporary file	hostname.nagios-plugins-argus.PDP.traffic.pickle
		–help	show this help message and exit	
		–port PORT	The port of the service	8152
		–url URL	The status endpoint URL of the service, e.g. https://hostname:port/status	
		–version	show program's version number and exit	
		–timeout TIMEOUT	The TCP timeout for the HTTPS connection in seconds	20
		–verbose	verbose mode	False





---

## Example of Authorization Requests and Policies

---

The following examples of policies and associated requests are meant to illustrate some of the possible policies, requests, and resulting authorization decisions.

Note, the examples below use the PAP's *simplified policy language* to express policies. So please review this syntax, before proceeding, if you are not yet familiar with it.

### 7.1 User Based Authorization

In this example one user is permitted, and one is denied, the ability to submit jobs on the CERN CE resource. Each user is identified by a X.509 subject DN. This example also shows how identity-based access control is simply a degenerate case of attribute-based access control.

#### 7.1.1 Authorization Policy

```
resource "http://cern.ch/authz/cel" {
  action "http://cern.ch/authz/actions/ce-submit" {
    rule permit {
      subject="CN=John Doe,OU=Standard Commercial Certificate,O=Acme,L=Zuerich,ST=Zuerich,C=CH"
    }
    rule deny {
      subject="CN=Jane Smith,OU=Standard Commercial Certificate,O=Acme,L=Zuerich,ST=Zuerich,C=CH"
    }
  }
}
```

#### 7.1.2 Requests

A request of:

- resource attributes:
  - **id:** urn:oasis:names:tc:xacml:1.0:resource:resource-id, **value:** http://cern.ch/authz/cel
  - any other attributes
- action attributes:

- **id:** urn:oasis:names:tc:xacml:1.0:action:action-id, **value:** http://cern.ch/authz/actions/ce-submit

- any other attributes

- subject attributes:

- **id:** urn:oasis:names:tc:xacml:1.0:subject:subject-id, **value:** CN=John Doe,OU=Standard Commercial Certificate,O=Acme,L=Zuerich,ST=Zuerich,C=CH

would result in an authorization decision of `permit` because the given resource and actions IDs match what was given in the policy and the subject ID DN matches what was given in the permit rule. If the request contained any other resource, action, or subject attribute these would be ignored since the policy does not reference them. If the request had not contained the attributes listed it would have resulted in an authorization decision of `not applicable` since no policy would have matched the request.

A request of:

- resource attributes:

- **id:** urn:oasis:names:tc:xacml:1.0:resource:resource-id, **value:** http://cern.ch/authz/cel

- any other attributes

- action attributes:

- **id:** urn:oasis:names:tc:xacml:1.0:action:action-id, **value:** http://cern.ch/authz/actions/ce-submit

- any other attributes

- subject attributes:

- **id:** urn:oasis:names:tc:xacml:1.0:subject:subject-id, **value:** CN=Jane Smith,OU=Standard Commercial Certificate,O=Acme,L=Zuerich,ST=Zuerich,C=CH

would result in an authorization decision of `deny` since the resource and action IDs match and this time the subject ID matches the requirement for the deny rule.

Note, in the second request, if the subject ID had contained at least the values `CN=John Doe,OU=Standard Commercial Certificate,O=Acme,L=Zuerich,ST=Zuerich,C=CH` and `CN=Jane Smith,OU=Standard Commercial Certificate,O=Acme,L=Zuerich,ST=Zuerich,C=CH` the authorization decision would have been `permit` since the permit rule would have been evaluated first and would have returned the result.

## 7.2 Per-VO Pilot Job Authorization Policy

This example shows a policy that allows pilot jobs from one VO, atlas, and denies pilot jobs from others. It also demonstrates rules that rely on more than one attribute.

### 7.2.1 Authorization Policy

```
resource "http://cern.ch/authz/cel" {
  action "http://cern.ch/authz/actions/ce-submit" {
    rule permit {
      vo="atlas"
      pilot-job="true"
    }
  }
}
```

```

    rule deny {
        pilot-job="true"
    }
}

```

## 7.2.2 Requests

A request of:

- resource attributes:
  - **id:** urn:oasis:names:tc:xacml:1.0:resource:resource-id, **value:** http://cern.ch/authz/cel
  - any other attributes
- action attributes:
  - **id:** urn:oasis:names:tc:xacml:1.0:action:action-id, **value:** http://cern.ch/authz/actions/ce-submit
  - any other attributes
- subject attributes:
  - **id:** http://authz-interop.org/xacml/subject/vo, **value:** atlas
  - **id:** http://example.org/authz/attribute/pilot-job, **value:** true

would result in a decision of `permit` since the atlas VO is permitted to submit pilot jobs. A request where the VO was `lhcb` or `cms` would result in a decision of `deny` since these VOs are not permitted to submit jobs.

A request of: \* resource attributes:

- **id:** urn:oasis:names:tc:xacml:1.0:resource:resource-id, **value:** http://cern.ch/authz/cel
- any other attributes \* action attributes:
  - **id:** urn:oasis:names:tc:xacml:1.0:action:action-id, **value:** http://cern.ch/authz/actions/ce-submit
- any other attributes \* subject attributes:
  - **id:** http://authz-interop.org/xacml/subject/vo, **value:** atlas

would result in a decision of `not applicable` since the current policy indicates that only jobs from the VO atlas that are also pilot jobs are permitted. Requests without a pilot-job attribute are not addressed by this policy.



---

## Legacy Pages from Twiki site

---

**Warning:** The pages below have been converted to the new documentation format without any specific review. They are coming from the former Argus and are probably obsolete. They are kept here for reference only.

```
%META:TOPICINFO{author="ad968f62f612332eff6b" date="1355496377" format="1.1" reprev="1.3" version="1.3"}% %META:TOPICPARENT{name="ArgusEMIDeployment"}%
```

### 8.1 Argus EMIR Publisher Configuration (EMI-3)

You can use [EMIR-SERP](#) to publish the Argus resource information to the [EMIR](#)Registy. EMIR-SERP uses the information already available in the resource BDII and publish it to an EMIR [DSR](#) endpoint.

#### 8.1.1 Local Resource BDII

Check that the local resource BDII is running and return service information:

```
ldapsearch -x -h localhost -p 2170 -b 'GLUE2GroupID=resource,o=glue'
objectClass=GLUE2Service
```

This query should return 1 entry for the Argus service. If it is not the case, something is fishy in the installtion...

#### 8.1.2 Install and Configure EMIR SERP

Install the EMIR SERP package:

```
yum install emir-serp
```

Edit the configuration file `/etc/emi/emir-serp/emir-serp.ini` and set the following:

```
...
url = http://emitbdsr1.cern.ch:9126
...
[servicesFromResourceBDII]
resource_bdii_url = ldap://localhost:2170/GLUE2GroupID=resource,o=glue
...
```

And start the EMIR-SERP service:

```
/sbin/service emir-serp start
```

You can verify that the information is correctly published by browsing: <http://emitbdsr1.cern.ch:9126/services>

```
%META:TOPICINFO{author="ad968f62f612332eff6b" date="1340980935" format="1.1" version="1.12"}%
%META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.2 Argus Services (EMI-1) Fine Tuning

### 8.2.1 Optimizing Argus Memory Usage

The **Argus 1.3** services (PAP, PDP and PEP Server) don't have fine tuned memory limits. For production sites it is therefore recommended to manually set a memory limit for each service.

**NOTICE:** Starting with **Argus 1.4**, the memory limits are already set, but you can adapt them for your special need.

#### Argus PAP Memory

In the PAP, the memory usage of the PAP depends directly on the number of policies.

- Default memory limit: 512 MB (v.1.4), 256MB (v.1.5)
  - No default memory limit set for version lower than v.1.4
- The default memory limit is typically enough for:
  - ~2'000 policies (resource - action - obligation - rule)
  - ~20'000 rules in one policy (~20'000 DN's or FQAN's)

#### PAP Memory Setting for Production

For production sites, which typically have less than 500 policies, we recommend to set the PAP memory limit to 256 MB

- **Recommended** memory limit (assuming less than 500 policies): 256 MB
- The value can be changed by setting the variable `PAP_JAVA_OPTS="-Xmx256m"` in the `/etc/sysconfig/argus-pap` configuration file

You have to restart the Argus PAP service to apply the new memory limit settings.

#### Argus PDP Memory

The PDP memory usage depends on the number of policies it has to import from the PAP.

- Default memory limit: 256 MB (v.1.4)
  - No default memory limit set for version lower than v.1.4
- The needed memory is comparable with the memory needed by the PAP

#### PDP Memory Setting for Production

For production sites, it is highly recommended to set a memory limit, to avoid a growing of the used memory (up to 1 GB) by the PDP.

- **Recommended** memory limit (assuming less than 500 policies in the PAP): 256 MB
- The value can be changed by setting the variable `PDP_JOPTS="-Xmx256M"` in the `/etc/sysconfig/argus-pdp` configuration file

You have to restart the Argus PDP service to apply the new memory limit settings.

## Argus PEP Server Memory

The PEP Server memory usage depends on the cache-size.

- Default memory limit: 512MB (v.1.3.1) and 128MB (v.1.4), and 256MB (v.1.5)
  - **No** default memory limit set for version lower than v.1.3.1
- Default value for the cache size: max. 500 cached PDP responses

The value for the cache size (`maximumCachedResponses`) can be changed following the PEP Server Configuration documentation.

### PEP Server Memory Setting for Production

The default memory limit set for the PEP Server v1.3.1 is still rather high, It can safely be lowered without any degradation of performance, and as long as the cache-size is not increased. For PEP Server version lower than v1.3.1, it is highly recommended to set a memory limit, to avoid a growing of the used memory (up to 1 GB).

- **Recommended** memory limit: 256 MB
  - The previous memory limit of 128MB in Argus 1.4.0 was too low for site with a lot of pool accounts.
- The value can be changed by setting the variable `PEPD_JOPTS="-Xmx256M"` in the `/etc/sysconfig/argus-pepd` configuration file

You have to restart the Argus PEP Server service to apply the new memory limit settings.

## 8.2.2 Production Logging Settings

By default the Argus 1.3 (EMI-1) services don't have production specific logging configuration, therefore, for production sites it is recommended to manually configure the logging for each service.

**NOTICE:** Starting with **Argus 1.4**, the services already use these production logging configuration.

For production sites, we have define the following logging policy:

- Log files rotation:
  - daily rotating (at midnight)
  - or when the file size  $\geq$  100MB
- Rotated log files are gzipped
- History of log files keep for 90 days

The Argus services uses the [Logback Logging Framework](#) to do their logging. The logback framework can be configured to implement the production logging policy.

### Argus PAP Logging

The Argus PAP service logging configuration is configured with the `/etc/argus/pap/logging/standalone/logback.xml` file.

#### PAP Logging Configuration for Production

To configure the Argus PAP service for production logging, please download the attached *configuration file* and store it as `/etc/argus/pap/logging/standalone/logback.xml`:

```
wget --no-check-certificate %ATTACHURL%/pap-logback.xml -O /etc/argus/pap/logging/standalone/logback.xml
```

You have to restart the PAP service to apply the new logging configuration.

## Argus PDP Logging

The Argus PDP service logging configuration is configured with the `/etc/argus/pdp/logging.xml` file.

### PDP Logging Configuration for Production

To configure the Argus PDP service for production logging, please download the attached *configuration file* and store it as `/etc/argus/pdp/logging.xml`:

```
wget --no-check-certificate %ATTACHURL%/pdp-logging.xml -O /etc/argus/pdp/logging.xml
```

The PDP will automatically reload its logging configuration, no need to restart the service.

## Argus PEP Server Logging

The Argus PEP Server service logging configuration is configured with the `/etc/argus/pepd/logging.xml` file.

### PEP Server Logging Configuration for Production

To configure the Argus PEP Server service for production logging, please download the attached *configuration file* and store it as `/etc/argus/pepd/logging.xml`:

```
wget --no-check-certificate %ATTACHURL%/pepd-logging.xml -O /etc/argus/pepd/logging.xml
```

The PEP Server will automatically reload its logging configuration, no need to restart the service.

```
%META:FILEATTACHMENT{name="pap-logback.xml" attachment="pap-logback.xml" attr="" comment="PAP logging config" date="1311669537" path="pap-logback.xml" size="2157" stream="pap-logback.xml" tmpFilename="/usr/tmp/CGItemp26785" user="ad968f62f612332eff6b" version="3"}%
%META:FILEATTACHMENT{name="pdp-logging.xml" attachment="pdp-logging.xml" attr="" comment="PDP logging config" date="1311669557" path="pdp-logging.xml" size="5541" stream="pdp-logging.xml" tmpFilename="/usr/tmp/CGItemp26834" user="ad968f62f612332eff6b" version="3"}%
%META:FILEATTACHMENT{name="pepd-logging.xml" attachment="pepd-logging.xml" attr="" comment="PEP Server logging config" date="1311669570" path="pepd-logging.xml" size="5067" stream="pepd-logging.xml" tmpFilename="/usr/tmp/CGItemp26858" user="ad968f62f612332eff6b" version="3"}%
%META:TOPICINFO{author="ad968f62f612332eff6b" date="1277121935" format="1.1" reprev="1.1" version="1.1"}%
%META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.3 Argus Development Tools

The tools used by the Argus PT to develop the Argus Authorization Service

### 8.3.1 Development

- Eclipse IDE for all Java and C development (<http://www.eclipse.org/>)
- CERN subversion server for the version control of all Argus components (<http://svnweb.cern.ch/world/wsvn/g1xa>)
- CERN wiki for the development and product documentation (<https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework>)



### 8.3.2 Build

- Maven for the compilation of the main Java components (<http://maven.apache.org/>)
- ant for the compilation of the PEP Java client library (<http://ant.apache.org/>)
- Autotools, configure and make for the compilation of the C components
- ETICS for the packaging and release management of all the Argus components (<https://etics.cern.ch/>)
- YAIM scripts for the glite-ARGUS node configuration

### 8.3.3 Test

- JUnit for some of the Java components
- Grinder for the internal load and stress testing of the server components (<http://grinder.sourceforge.net/>)

### 8.3.4 QA

- GGUS for users support (<http://www.ggus.org>)
- LCG Savannah for bugs, tasks and patches tracking (<https://savannah.cern.ch>)
- A code review was done by the PSNC Security Team, but we don't have enough resource within the Argus PT to do it ourself
- CERN vNode test bed using the automatic YAIM-gen for product certification

### 8.3.5 Infrastructure and Repository

- Some virtual machines for internal testing
- CERN YUM repository for deployment
- YAIM for the glite-ARGUS node configuration

```
%META:TOPICINFO{author="ad968f62f612332eff6b" date="1271254649" format="1.1" reprev="1.1" version="1.1"}% %META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.4 GGUS User Support for Argus

### 8.4.1 Argus Support Unit FAQ

#### 1801 What is the purpose of the Argus Support?

Provide support for site administrators on the installation, the configuration and the operation of the Argus Authorization Service.

#### 1802 Who is responsible for Argus Support?

Argus product team (PT) members.

### 1803 How does a ticket arrive at Argus Support?

Currently through the existing `argus-support@cern.ch` mailing list.

### 1804 What does the Argus Support manager have to do?

Dispatch support request.

### 1805 What does the person dealing with Argus Support have to do?

- Try to solve the problem with the user.
- If necessary, submit a bug in savannah.
- If relevant, update our FAQ with problem and solution.

### 1806 What documentation is available on Argus Support?

Extensive documentation is available at <https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework>

```
%META:TOPICINFO{author="ad968f62f612332eff6b" date="1254385015" format="1.1" version="1.3"}%  
%META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.5 Authorization Service: Load and Lifetime Testing

It is important to perform load and lifetime tests on any service being put in to production. Load tests are used to determine how the service behaves, and eventually fails, under load while lifetime tests determine how the service behaves over a prolonged period of time with average load.

Note, this testing framework used here was actually developed by the [Shibboleth](#) team for testing their software but it allows other test packs to be installed so it can also be used in testing this service.

### 8.5.1 Testing Framework Basics

#### Requirements

- Unix OS based machine(s)
- [Subversion](#) client (necessary in order to download the framework)
- Java 1.5

#### Downloading and Installing the Framework

1. Checkout the base testing framework. The resulting `grinder-framework` directory will be known as `GF_HOME` throughout the rest of this document.

```
• svn co https://svn.middleware.georgetown.edu/shib-extension/java-loadtest/trunk/grinder-framework
```

2. Within `_GFHOME/test` checkout the Argus test packs

- `svn co http://svnweb.cern.ch/guest/glxa/loadtests/pepcli-authz pepcli-authz`
- `svn co http://svnweb.cern.ch/guest/glxa/loadtests/pepcli-multiauthz pepcli-multiauthz`
- `svn co http://svnweb.cern.ch/guest/glxa/loadtests/pepj-authz pepj-authz`

Installation is now complete.

## Running the Framework

The load testing framework is composed of two parts, an agent and a console. The agent performs the actual work while the console, a GUI application, collects the results from agents and displays some metrics based on those results

To run the console execute the `GF_HOME/bin/console.sh` command. This will start the console which will listen on port 6372 for agent results. Be sure that this port is accessible to the agents you start.

To run the console execute the `GF_HOME/bin/agent.sh test_name` command. The name of the test is dependent on which test you will run and is given in the documentation for those tests. You can install agents on more than one machine and have them report back to a single client. This is useful if one agent machine is not capable of generating enough load.

## Common Configuration Options

Each test package contains a `test.properties` file that contains configuration information for the given test. The later half of these options are Grinder specific properties. In most cases you'll need to change the following properties.

Property Name	Value
<code>grinder.consoleHost</code>	Host IP address of the machine on which the console is running
<code>grinder.processes</code>	Number of grinder processes to run. Should be equal to the number of CPU cores on the agent machine.
<code>grinder.threads</code>	Number of unique clients, per process, that will connect to the service
<code>grinder.runs</code>	Number of times each client will connect to the service, 0 (zero) means to keep going until the console indicates the test should be stopped. Using a value of 1 (one) is useful for testing that your configuration is correct.

### 8.5.2 PEPCLI Test

This test uses the C language PEP CLI to issue basic authorization requests. It a good test to run if you are trying get measurements for applications which do not maintain state between requests (e.g. a new client is created for every request).

**Test Name:** `pepcli-authz`

## Configuration Options

Property Name	Value	Required
pepcli.bin	The path to the pepcli binary.	Yes
pepcli.pepds	A space separated list of PEPd endpoints	Yes
pepcli.certchain	Path, relative to GF_HOME, to the certchain used to identify the subject or the request	Yes
pepcli.resourceid	The resource ID used within the request	One is required.
pepcli.resourceid.prefix	Prefix added to the <i>grinder-agent-name</i> to create the resource ID used in the request	^
pepcli.actionid	The action ID used within the request	One is required.
pepcli.actionid.prefix	Prefix added to the <i>grinder-agent-run-number</i> to create the resource ID used in the request	^
pepcli.timeout	The connection timeout in seconds	No, defaults to 30

### 8.5.3 PEPCLI Test

This test uses the C language PEP CLI to issue multiple basic authorization requests. It a good test to run if you are trying get measurements for applications which do not maintain state between requests (e.g. a new client is created for every request).

**Test Name:** pepcli-multiauthz

## Configuration Options

Property Name	Value	Required
pepcli.bin	The path to the pepcli binary.	Yes
pepcli.pepds	A space separated list of PEPd endpoints	Yes
pepcli.timeout	The connection timeout in seconds	No, defaults to 30
pepcli.tests	A set of test inputs. See below for format.	Yes

## Test Input Format

The test import format is as follows:

```
[ \
  ['ce1', 'submit', '/home/jsmith/.globus/usercert.pem', 'Permit'], \
  ['ce2', 'submit', '/home/jsmith/.globus/usercert.pem', 'Deny'], \
  ['ce3', 'submit', '/home/jsmith/.globus/usercert.pem', 'Not Applicable'], \
]
```

The property value is contained in matching '[' and ']' brackets. Then each test individual test scenario is contained with matching '[' and ']'. Each testing scenario has four comma-separated values: the resource ID, action ID, and certificate chain used in the request and then the expected outcome. Each test scenario is also comma-separated. The use of the '\ ' character can be used to indicate that the property value continues on to the next line.

### 8.5.4 PEP-J Test

This test uses the Java language PEP CLI to issue authorization requests. It is a good test to run if you are trying to get measurements for application which maintain state between requests (e.g. a single client is used for multiple requests).

**Test Name:** pepj-authz

#### Configuration Option

**Property Name** | **Value** | **Required** | | authz.client.config | Path to the client configuration file. Relative paths are relative to GF\_HOME. | Yes |

```
%META:TOPICINFO{author="joel_2ecasutt_40cern_2ech" date="1320850921" format="1.1" version="1.7"}%
%META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.6 Authorization Services Testing Summary

### 8.6.1 Hardware Description

For the performance and aging tests, we installed:

- EMI-1 Argus 1.3 (PAP, PDP and PEP Server):
  - 1 Xeon CPU 2.33GHz (Dual Core)
  - 5 GB RAM
- PEP client pepcli:
  - 4 hosts running the Grinder framework

Update: Results for the 1.3.1 update using the same Hardware

There is a significant drop in memory consumption coming with the 1.3.1 update for the pepd and the pdp server. Find some load test using those Versions below.

### 8.6.2 Performance Tests

The performance testing of the Authorization Services was done using the hardware described above. The setup of the test was basically the same as described [here](#), using the following options. Each agent (host running grinder-framework) opens two worker, and each worker opens 5 threads. Therefore a maximum of 40 (4\*2\*5) simultaneous requests are achieved.

#### Test Description (SSL enabled)

This is the most common case, since the default configuration with YAIM results in a PEPd with SSL enabled

On the Argus Server

The default configuration was used, only the cache-size was lowered to 200 requests

- pepd.ini: maximumCachedResponses = 200

The obligations needed for this test were added to the PAP

- pap-admin ap permit subject="CN=Joel Casutt D88E5396,O=SWITCH,DC=slcs,DC=switch,DC=ch" -action "submit" -resource "cel\_.\*"

- pap-admin ap deny subject="CN=Joel Casutt D88E5396,O=SWITCH,DC=slcs,DC=switch,DC=ch" -action "submit" -resource "ce2\_.\*"

### The Test-Scripts

To assure that not only cached requests are used by the PEPd, but that the PEPd also sends requests to the PDP, more than 200 different requests need to be defined in the properties file. This can easily be done by changing the resource (e.g. ce1\_100, ce1\_101, ...)

### Results and Metrics

- Test ran for about 16h without interruption
- 1'698'886 authorization requests processed
- 30 requests per second sustained
- 1340 ms average round trip time (pepcli -> PEPd (-> PDP -> PEPd) -> pepcli)
- Memory usage of the PEPd during the load-test:   
 
- Memory usage of the PDP during the load-test:   
 

### Results and Metrics for the 1.3.1-update

- Test ran for about 63h without interruption
- ~6.9 M authorization requests processed
- 30 requests per second sustained
- 900 ms average round trip time (pepcli -> PEPd (-> PDP -> PEPd) -> pepcli)
- Requests per second during the load test for the 1.3.1 update:   
 
- Memory usage of the pdp during the load test for the 1.3.1 update:   
 
- Memory usage of the pepd during the load test for the 1.3.1 update:   
 

## 8.6.3 Aging Tests

This test was done analogue to the performance test, but with only one Agent starting one worker starting one thread. This resulted in about 10 requests per second.

### Test Description

The test was done using the same options as the performance test.

### Results and Metrics

- Test ran for about 43h without interruption
- 1'486'711 authorization requests processed

- 10 requests per second sustained
- 101 ms average round trip time (pepcli -> PEPd (-> PDP -> PEPd) -> pepcli)
- Memory usage of the PDP during the long-test:   
 
- Memory usage of the PEPd during the long-test:   
 

```
%META:TOPICINFO{author="joel_2ecasutt_40cern_2ech" date="1322576072" format="1.1" version="1.4"}%
%META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.7 Authorization Services Testing Summary

### 8.7.1 Hardware Description

For the performance and aging tests, we installed:

- EMI-1 Argus 1.4.0 (PAP, PDP and PEP Server):
  - 1 Xeon CPU 2.33GHz (Dual Core)
  - 5 GB RAM
- PEP client `pepcli`:
  - 3 hosts running the Grinder framework

### 8.7.2 Performance Tests

The performance testing of the Authorization Services was done using the hardware described above. The setup of the test was basically the same as described [here](#), using the following options. Each agent (host running grinder-framework) opens two worker, and each worker opens 2 threads. Therefore a maximum of 12 (3\*2\*2) simultaneous requests are achieved.

#### Test Description (SSL enabled)

This is the most common case, since the default configuration with YAIM results in a PEPd with SSL enabled

On the Argus Server

The default configuration was used The policies needed for this test were added to the PAP

- `pap-admin ap permit pfqan="/dteam" -action "submit" -resource "ce1_.*"`
- `pap-admin ap deny pfqan="/dteam" -action "submit" -resource "ce2_.*"`

while the star denotes a number between 1 and 500, so in total the pap was loaded with 1'000 policies each one containing a rule and a obligation

The Test-Scripts

To assure that not only cached requests are used by the PEPd, but that the PEPd also sends requests to the PDP, more than 200 different sets of Action/Resource/Obligation have been defined in the properties file. This can easily be done by changing the resource (e.g. `ce1_100`, `ce1_101`, ...). More than that 100 certificates issued by the [ARC instant online CA](#) were randomly combined with those sets, leading to roughly 1'000 different possible requests.

Results and Metrics

- Test ran for about 26.5h without interruption
- 4'078'768 authorization requests processed
- 43 requests per second sustained
- 274 ms average round trip time (pepcli -> PEPd (-> PDP -> PEPd) -> pepcli)
- Memory consumption of the 1.4.0 pdp during a load-test:   
 <br /> 
- Memory consumption of the 1.4.0 pepd during a load-test:   
 <br /> 
- Average requests per second during the load-test:   
 <br /> 

### 8.7.3 Aging Tests

This test was done analogue to the performance test, but with only one Agent starting one worker starting one thread. This resulted in about 9 requests per second.

#### Test Description

The test was done using the same options as the performance test.

#### Results and Metrics

- Test ran for about 140h without interruption
- 4'951'677 authorization requests processed
- 9 requests per second sustained
- 97.5 ms average round trip time (pepcli -> PEPd (-> PDP -> PEPd) -> pepcli)
- Memory consumption of the 1.4.0 pdp during a aging-test:   
 <br /> 
- Memory consumption of the 1.4.0 pepd during a aging-test:   
 <br /> 
- Average requests per second during the aging-test:   
 <br /> 

%META:TOPICINFO{author="joel\_2ecasutt\_40cern\_2ech" date="1322731035" format="1.1" reprev="1.2" version="1.2"}% %META:TOPICPARENT{name="AuthorizationFramework"}%



## 8.8 Authorization Services Testing Summary

### 8.8.1 Hardware Description

For the performance and aging tests, we installed:

- EMI-1 Argus 1.4.0 (PAP, PDP and PEP Server):
  - 1 QEMU Virtual CPU 2.66GHz
  - 384 MB RAM
- PEP client `pepcli`:
  - 3 hosts running the Grinder framework

### 8.8.2 Performance Tests

The performance testing of the Authorization Services was done using the hardware described above. The setup of the test was basically the same as described [here](#), using the following options. Each agent (host running grinder-framework) opens two worker, and each worker opens 2 threads. Therefore a maximum of 12 (3\*2\*2) simultaneous requests are achieved.

#### Test Description (SSL enabled)

This is the most common case, since the default configuration with YAIM results in a PEPd with SSL enabled

On the Argus Server

The default configuration was used The policies needed for this test were added to the PAP


- `pap-admin ap permit pfqan="/dteam" -action "submit" -resource "ce1_.*"`
- `pap-admin ap deny pfqan="/dteam" -action "submit" -resource "ce2_.*"`

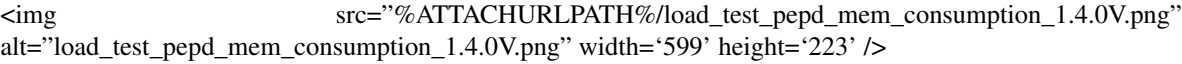
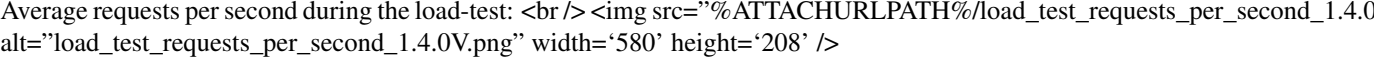
while the star denotes a number between 1 and 500, so in total the pap was loaded with 1'000 policies each one containing a rule and a obligation

The Test-Scripts

To assure that not only cached requests are used by the PEPd, but that the PEPd also sends requests to the PDP, more than 200 different sets of Action/Resource/Obligation have been defined in the properties file. This can easily be done by changing the resource (e.g. `ce1_100`, `ce1_101`, ...). More than that 100 certificates issued by the [ARC instant online CA](#) were randomly combined with those sets, leading to roughly 1'000 different possible requests.

Results and Metrics

- Test ran for about 42h without interruption
- 4'362'423 authorization requests processed
- 28.9 requests per second sustained
- 409 ms average round trip time (`pepcli -> PEPd (-> PDP -> PEPd) -> pepcli`)
- Memory consumption of the 1.4.0 pdp during a load-test:   

  
`src="%ATTACHURLPATH%/load_test_pdp_mem_consumption_1.4.0V.png"`  
`alt="load_test_pdp_mem_consumption_1.4.0V.png" width="598" height="223" />`

- Memory consumption of the 1.4.0 pepd during a load-test:   
 width='599' height='223' />
- Average requests per second during the load-test:   
 width='580' height='208' />

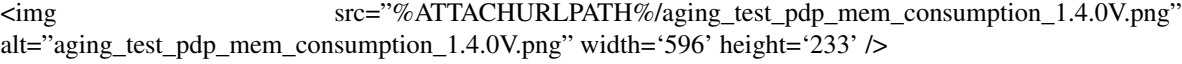
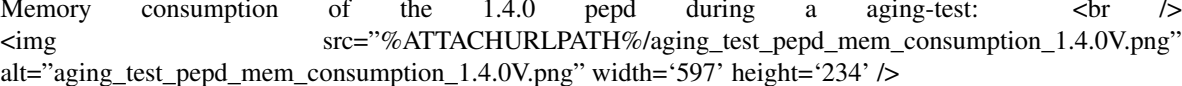
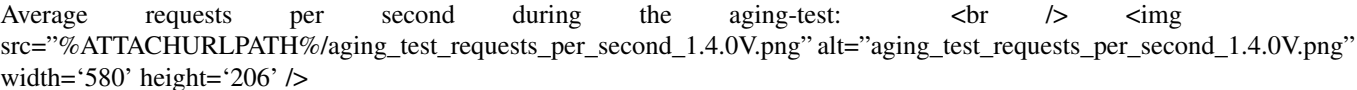
### 8.8.3 Aging Tests

This test was done analogue to the performance test, but with only one Agent starting one worker starting one thread. This resulted in about 9 requests per second.

#### Test Description

The test was done using the same options as the performance test.

#### Results and Metrics

- Test ran for about 190h without interruption
- 7'332'606 authorization requests processed
- 10.7 requests per second sustained
- 89.1 ms average round trip time (pepcli -> PEPd (-> PDP -> PEPd) -> pepcli)
- Memory consumption of the 1.4.0 pdp during a aging-test:   
 width='596' height='233' />
- Memory consumption of the 1.4.0 pepd during a aging-test:   
 width='597' height='234' />
- Average requests per second during the aging-test:   
 width='580' height='206' />

```
%META:TOPICINFO{author="ad968f62f612332eff6b" date="1257252646" format="1.1" reprev="1.2" version="1.2"}% %META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.9 Argus: Quick Start: Manual Installation

### 8.9.1 Prerequisites

In order to run the Argus service you will need the following:

- a working Java 5 or better installation. Use `java -version` to determine your Java version.
- a recent version of openssl (>0.9.7) installed on your system
- LCG trusted Certificate Authorities certificates installed. In SL4 and SL5, the `lcg-CA` package provides those certificates.
- a valid, PEM encoded X509 server certificate issued by a trusted CA.

## 8.9.2 Installation

The Argus service is composed of three components, the PAP, PDP, and PEPd. The following steps provide instructions for installing each component.

### PAP Installation

1 **Download** the latest version version of the software. 1 Expand the downloaded archive to the `/opt/argus` directory.

- This resulting installation directory `/opt/argus/pap` will be referred to, in the documentation, as `PAP_HOME` 1 Export the environment variable `PAP_HOME=/opt/argus/pap` 1 Add the `/opt/argus/pap/bin` to your `PATH` in order to use the `PAP pap-admin` command

### PDP Installation

1. **Download** the latest version version of the software. 1 Expand the downloaded archive in to the `/opt/argus` directory.

- This resulting installation directory `/opt/argus/pdp` will be referred to, in documentation, as `PDP_HOME`

### PEPd Installation

1 **Download** the latest version version of the software. 1 Expand the downloaded archive in to the `/opt/argus` directory.

- The resulting installation directory `/opt/argus/pepd` will be referred to, in documentation, as `PEPD_HOME`

## 8.9.3 Starting/Stop the Services

To start/stop the PAP you can use the command `/opt/argus/pap/bin/pap-standalone start|stop`.

To start/stop the PDP you can use the command `/opt/argus/pdp/bin/pdpctl.sh start|stop`.

To start/stop the PEPd you can use the command `/opt/argus/pepd/bin/pepdctl.sh start|stop`.

You should always start the PAP before the PDP or else the PDP will not be to retrieve its policy from the PAP.

---

Installation is now complete, proceed to configuring the Argus service.

```
%META:TOPICINFO{author="ad968f62f612332eff6b" date="1271340916" format="1.1" version="1.7"}%
%META:TOPICPARENT{name="AuthzQSYumYaimInstall"}%
```

## 8.10 Argus: Quick Start: Site Policy Setup

### 8.10.1 Policy Basics

A site's policies are maintained using the command `/opt/argus/pap/bin/pap-admin`. Initially the PAP contains an empty policy for the site so no one will be permitted to do anything. The command `pap-admin list-policies` lists policies, in the simplified policy language notation, currently active in the PAP.

The policies you add will be evaluated from most to least recent and the first policy that matches is the result returned by Argus. So, if you added a policy that would deny a user and then added one that would permit the user the result of an authorization request will be permit since the permit policy is most recent. While not covered here, the exhaustive PAP command line interface reference provides information for how to re-order policies.

To begin, you will need to create permit policies, for individuals users, VOs, and FQANs, which reflect your site's access policy. Each of the commands listed require a resource ID and action ID which are deployment and application specific, respectively. These ID were described in the introduction to the Argus system. For the purposes of this quick-start we'll use `http://example.org` as the resource ID and `http://example.org/action` as the action ID. At this point you do not need to worry about what the correct values would be for a production deployment.

**NOTE**, as you change policies you will need to restart the PDP in order to force it to reread the policy from the PAP. It would do this by default but only after a couple of hours. Restarting forces it do this immediately.

### Permit by User

Command: `pap-admin add-policy permit --resource "RESOURCE_ID" --action "ACTION_ID" dn="USER_DN"`

The provided `USER_DN` must be in the RFC2253 standard form, not the proprietary format used by OpenSSL by default. To get the standard form of the DN you can use the command `openssl x509 -noout -nameopt RFC2253 -subject -in CERT_PATH`

Here's an example that permits a user identified by their DN: `pap-admin add-policy permit --resource "http://example.org" --action "http://example.org/action" dn="CN=John Smith,OU=Standard Commercial Certificate,O=SWITCH,L=Zuerich,ST=Zuerich,C=CH"`

### Permit by VO

Command: `pap-admin add-policy permit --resource "RESOURCE_ID" --action "ACTION_ID" vo="VO"`

The name of a VO is the string that appears between, but not including, the first two forward slashes ('/') of an FQAN.

Here's an example that permits a user identified by their VO: `pap-admin add-policy permit --resource "http://example.org" --action "http://example.org/action" vo="atlas"`

### Permit by Primary FQAN

Command: `pap-admin add-policy permit --resource "RESOURCE_ID" --action "ACTION_ID" pfqan="FQAN"`

The provided FQAN must be in full canonical (long) form (e.g. `/atlas/Role=Production/Capability=NULL` or `/atlas/Role=NULL/Capability=NULL`). You may use [Java regular expression](#) in this string.

Here's an example that permits a user identified by their primary FQAN: `pap-admin add-policy permit --resource "http://example.org" --action "http://example.org/action" pfqan="/atlas/Role=Production/.+"`

### Permit by Secondary FQAN

Command: `pap-admin add-policy permit --resource "RESOURCE_ID" --action "ACTION_ID" fqan="FQAN"`

The provided FQAN must be in full canonical (long) form (e.g. `/atlas/Role=Production/Capability=NULL` or `/atlas/Role=NULL/Capability=NULL`). You may use [Java regular expression](#) in this string.

Here's an example that permits a user identified by their secondary FQAN: `pap-admin add-policy permit --resource "http://example.org" --action "http://example.org/action" pfqan="/atlas/higgs/Role=NULL/Capability=NULL"`

## Explicit Banning

At times you may wish to deny a specific user, FQAN, or VO. As you may have guessed by looking at the commands above, you can create deny policies by changing the `permit` argument to `deny`.

Here are some examples:

```
pap-admin add-policy deny --resource "http://example.org" --action
"http://example.org/action" pfqan="/atlas/higgs/Role=NULL/Capability=NULL"

pap-admin add-policy deny --resource "http://example.org" --action
"http://example.org/action" vo="cms"
```

## Removing Policies

The command `pap-admin remove-policy POLICY_ID` will remove an existing policy from the PAP. The `POLICY_ID` is the ID for the policy shown by the `pap-admin list-policies --show-all-ids` command.

### 8.10.2 Loading and Testing Policies

Now that you have some basic policy commands you should experiment with creating and testing policies.

To test the policies you can use the command `pepcli`. This debugging tool can be used in two different ways. First you can supply it the cert of a user or you can mock up a request by specifying a subject DN and primary and second FQANs. See the `pepcli` documentation for complete list of options or use the examples below.

This command provides the result that the user, represented by the given certificate, would receive: `pepcli --pepd !http://127.0.0.1:8154/authz --resourceid "http://example.org" --actionid "http://example.org/action" --certchain CERT_PATH`. The certificate may be an end-entity certificate, a plain proxy certificate, or a proxy certificate with VOMS extensions.

This command allows you to create a request containing a specific user DN, primary FQAN (the first one listed), and secondary FQANs (subsequent ones listed): `pepcli --pepd !http://127.0.0.1:8154/authz --resourceid "http://example.org" --actionid "http://example.org/action" --subjectid USER_DN --fqan PRIMARY_FQAN --fqan SECONDARY_FQAN --fqan SECONDARY_FQAN`

So, for example add a policy to allow people with the primary FQAN of `/atlas/higgs/Role=NULL/Capability=NULL` to be permitted using the following command `pap-admin add-policy permit --resource "http://example.org" --action "http://example.org/action" pfqan="/atlas/higgs/Role=NULL/Capability=NULL"`.

Then test the policy using the `pepcli` command `pepcli --pepd !http://127.0.0.1:8154/authz --resourceid "http://example.org" --actionid "http://example.org/action" --subjectid "CN=John Smith,OU=Standard Commercial Certificate,O=SWITCH,L=Zuerich,ST=Zuerich" --fqan "/atlas/higgs/Role=NULL/Capability=NULL"`. The result should be a permit.

### 8.10.3 Cleaning Up

Before you go further be sure to clean out of your test policies from the PAP. You do this by removing select policies as shown above or you may use the command `pap-admin remove-all-policies` to remove all policies from

the PAP. Be sure to restart your PDP as well so that it picks up your new policy set.

— You are now done with this quick-start guide. At this point you should have a functioning Argus service and be able to add/remove basic policies from your site. It is recommended that you

```
%META:TOPICINFO{author="chad_2elajoie_40switch_2ech" date="1257249650" format="1.1" version="1.1"}%  
%META:TOPICPARENT{name="AuthzQSMInstall"}%
```

## 8.11 Argus: Quick Start: Manual Configuration

*TODO*

---

Configuration is now complete, proceed to setting up your site policies.

```
%META:TOPICINFO{author="joel_2ecasutt_40cern_2ech" date="1311164000" format="1.1" reprev="1.17" version="1.17"}%  
%META:TOPICPARENT{name="AuthorizationFramework"}%
```

## 8.12 Quick Start: glite 3.2 Argus Installation

### 8.12.1 Installation and Configuration

The PAP, PDP, and PEPd are bundled together as the gLite node type `glite-ARGUS`.

#### YUM Repository

Copy the <http://grid-deployment.web.cern.ch/grid-deployment/glite/repos/3.2/glite-ARGUS.repo> repository file into your local `/etc/yum.repos.d` directory.

#### Packages Installation

To install and initially configure the service do the following: 1 Install the EUGridPMA trust anchors bundle:  
`yum install lcg-CA` 1 If not already installed, install the Java VM OpenJDK 1.6:  
`yum install java-1.6.0-openjdk` 1 Install the glite-ARGUS metapackage:  
`yum install glite-ARGUS` 1 In your `YAIM site-info.def` configuration set at least the following variables:

- the `ARGUS_HOST` property to the hostname of the machine on which you are installing
- the `PAP_ADMIN_DN` property to the subject DN of your user certificate. Allow the use the `pap-admin` command
- the `USERS_CONF`, `GROUP_CONF`, and `VOS` property to their appropriate values for your site 1 Configure the `ARGUS_server` node type with YAIM:  
`yaim -c -s site-info.def -n ARGUS_server`

#### Example Configuration

Example `site-info.def` for the `ARGUS_server` node type:

```
#
# glite-ARGUS hostname
#
ARGUS_HOST=vesta.switch.ch

# users and groups
USERS_CONF=/opt/glite/yaim/etc/users.conf
GROUPS_CONF=/opt/glite/yaim/etc/groups.conf

# Supported VOs
VOS="dteam"

VO_DTEAM_VOMS_CA_DN="'/DC=ch/DC=cern/CN=CERN Trusted Certification Authority' '/DC=ch/DC=cern/CN=CERN
VO_DTEAM_VOMSES="'dteam lcg-voms.cern.ch 15004 /DC=ch/DC=cern/OU=computers/CN=lcg-voms.cern.ch dteam

# PAP administrator DN
PAP_ADMIN_DN="/DC=org/DC=acme/CN=John Doe"
```

At this point, you should have a running PAP, PDP, and PEP daemon.

## 8.12.2 Operation Environment

You should now have Argus installed.

### Directories

There are a few directories that you should be aware of:

- `/opt/argus/pap` contains the PAP component of Argus
- `/opt/argus/pdp` contains the PDP component of Argus
- `/opt/argus/pepd` contains the PEPd component of Argus
- `/etc/grid-security/certificates` contains the CAs trusted by Argus
- `/etc/grid-security/grid-mapfile` contains the mappings from DN/FQANs to local user accounts
- `/etc/grid-security/groupmapfile` contains the mappings from FQANs to local groups
- `/etc/grid-security/gridmapdir` contains grid user to local user account mappings
- `/etc/grid-security/vomsdir` contains the VOMS servers trusted by Argus

Note, if you have an existing site and wish to use your existing account mappings you can move your existing mappings over to the Argus host. Simply tar your existing mappings (`tar -cf gridmap.tar /etc/grid-security/gridmapdir`) and then transfer them over the Argus host and untar them (`tar -xf gridmap.tar`).

### Endpoint URL

Argus is a network service. Like other network services you should ensure that your network settings only allow anticipated clients communicate with the service.

- Argus 1.1 enables by default client authentication on the service endpoint, therefore the endpoint URL scheme is now **HTTPS**.
- Client applications must be able to contact the service port **8154**

- Example Argus 1.1 endpoint URL (with client authN): `https://argus.example.org:8154/authz`
- All other ports (8150-8153) should only be accessible within the Argus service host itself.

### 8.12.3 Starting/Stopping the Services

To start/stop the PAP you can use the command `/etc/init.d/pap-standalone start|stop`.

To start/stop the PDP you can use the command `/etc/init.d/pdp start|stop`. You can also force a reload of the policies, retrieved from the PAP, with `/etc/init.d/pdp reloadpolicy`.

To start/stop the PEPd you can use the command `/etc/init.d/pepd start|stop`. You can also clear the responses cache with `/etc/init.d/pepd clearcache`.

You should always start the PAP before the PDP or else the PDP will not be to retrieve its policy from the PAP.

---

Installation is now complete, proceed to setting up your site policies.

### 8.12.4 Global Banning Configuration

The following are the required steps to make your Argus server import the global grid banning policies. These policies are maintained by OSCT / EGI CSIRT. The server is located at CERN.

Notice that in following the steps below you will be trusting the global banning server and its policies, so that e.g. users banned by the OSCT / EGI CSIRT will also be rejected at your site as a result.

The `pap-admin` command is normally installed in `/opt/argus/pap/bin`

Add the Global Banning PAP server from CERN as a remote PAP:

```
pap-admin add-pap centralbanning argus.cern.ch "/DC=ch/DC=cern/OU=computers/CN=argus.cern.ch"
```

The CERN PAP is now listed but is by default still disabled.

```
pap-admin enable-pap centralbanning
```

The CERN PAP is now enabled.

```
pap-admin set-paps-order centralbanning default
```

The CERN PAP policies are now parsed before the local policies, so that e.g. a user banned by OSCT is immediately rejected. This step is important as only with this order can black listing work.

```
pap-admin refresh-cache centralbanning
```

The local pap cache, comprising also the CERN policies, is refreshed and the new policies are made available. The policies are then fetched automatically by the server every polling interval seconds or manually when the `refresh-cache` command is sent to the server.

```
%META:TOPICMOVED{by="ad968f62f612332eff6b" date="1271340916" from="EGEE.AuthzQSYaimInstall" to="EGEE.AuthzQSYumYaimInstall"}%
```



---

## Argus Service Deployment for EMI

---

Argus Service Reference Card: <https://twiki.cern.ch/twiki/bin/view/EMI/ArgusSRC>

### 9.1 Requirements

#### 9.1.1 Platform

- For **EMI-3** the supported platforms are **Debian6/x86\_64**, and **SL5/x86\_64** and **SL6/x86\_64** with the [EPEL](#) repository **enabled**.
- For **EMI-2** the supported platforms are **SL5/x86\_64** and **SL6/x86\_64** with the [EPEL](#) repository **enabled**.
- For **EMI-1** the supported platform is **SL5/x86\_64** with the [EPEL](#) repository **enabled**.

#### 9.1.2 Host Certificate

The Argus services require a valid host certificate. The key pair is typically installed in `/etc/grid-security/hostcert.pem` and `/etc/grid-security/hostkey.pem`

### 9.2 Installation with YUM

The Argus Authorization Service is a bundle of 3 services (PAP, PDP and PEP Server) and is available in the EMI repository. The Argus metapackage should be installed with YUM.

#### 9.2.1 Install the EGI IGTF Bundle

If not already present on your host, install the EGI IGTF trust anchors. More information available online: [https://wiki.egi.eu/wiki/EGI\\_IGTF\\_Release](https://wiki.egi.eu/wiki/EGI_IGTF_Release)

- Install the EGI IGTF trust anchors with YUM

#### 9.2.2 Install fetch-crl

If not already present on your host, install the `fetch-crl` cron job (from the [EPEL](#) repository), run it once, and enable the cron job:

```
# install
yum install fetch-crl
# run it immediately (it can take some time...)
/usr/sbin/fetch-crl
# enable the periodic fetch-crl cron job
/sbin/chkconfig fetch-crl-cron on
/sbin/service fetch-crl-cron start
```

### 9.2.3 Install the EMI Repository

If not already configured on your host, install the EMI repositories package as described in the EMI generic installation and configuration guide for [EMI 1](#) or [EMI 2](#), or [EMI 3](#).

And update the YUM cache: `yum makecache`

### 9.2.4 Install the Argus Metapackage

The `emi-argus` metapackage bundles the 3 Argus services: PAP, PDP and PEP Server

Therefore, the `emi-argus` metapackage is the simplest way to install the Argus Authorization Services (PAP, PDP and PEP Server) on your host.

Use YUM to install the Argus metapackage: `yum install emi-argus`

At this point all 3 Argus services (PAP, PDP and PEP server) are installed on your host. You must now continue with the Argus service configuration with YAIM.

## 9.3 Update with YUM

Argus and its components are packages in the EMI and UMD repositories, and therefore automatically updated if a new version is available and `yum update` is executed. The updating process however is stopping the Argus services (PAP, PEPd and PDP). To get Argus back to work after an update it is recommended to rerun Yaim and/or restart the services.

**Please make sure that your system does not use the automated yum updating service, since this may lead to a stopped Argus server in case of a unnoticed update.**

## 9.4 Configuration with YAIM

The `ARGUS_server` node type is available to configure the Argus service with YAIM.

### 9.4.1 Argus YAIM Configuration Variables

Description of all the available Argus YAIM configuration variables: <https://twiki.cern.ch/twiki/bin/view/EGEE/ArgusEMIIaimConfigur>

### 9.4.2 Mandatory YAIM Variables

- `ARGUS_HOST` Fully qualified host name (FQHN) of the Argus host

- PAP\_ADMIN\_DN Certificate distinguished name (DN) of the administrator, allowed to use the `pap-admin` command
- SITE\_NAME BDII site name
- USERS\_CONF Absolute location of the users configuration file
- GROUPS\_CONF Absolute location of the groups configuration file
- VOS List of supported VO names
- VO\_<vo-name>\_VOMS\_CA\_DN VOMS CA DN for each VO name listed in VOS
- VO\_<vo-name>\_VOMSES VOMS definition for each VO name listed in VOS

The `USERS_CONF` and `GROUPS_CONF` configuration files **MUST** be the same on the Argus host as on the client host (CREAM, WMS, gLExec, ...). On successful authorization, the Argus PEP Server is configured to determine the user/group mapping (pool account) for this authorization and send it the client. Therefore, the client must be able to map the resulting user mapping received with the authorization decision.

### 9.4.3 Argus `site-info.def` Configuration

Your `site-info.def` for Argus must contain at least the following variable:

```
# BDII site name
SITE_NAME=MySiteName

# Argus service hostname
ARGUS_HOST=argus.example.org

# PAP administrator DN allowed to use 'pap-admin' command
PAP_ADMIN_DN="/DC=org/DC=acme/CN=John Doe"

# Users and Groups definition for grid and group mapfile
USERS_CONF=/opt/glite/yaim/examples/users.conf
GROUPS_CONF=/opt/glite/yaim/examples/groups.conf

# Supported VOs
VOS="dteam"

VO_DTEAM_VOMSES="'dteam voms.hellasgrid.gr 15004 /C=GR/O=HellasGrid/OU=hellasgrid.gr/CN=voms.hellasgrid.gr'"
VO_DTEAM_VOMS_CA_DN="'/C=GR/O=HellasGrid/OU=Certification Authorities/CN=HellasGrid CA 2006' '/C=GR/O=HellasGrid/OU=HellasGrid'"
```

See the documentation of all the supported Argus YAIM *configuration variables* <[argus\\_emi\\_yaim\\_configuration](#)>.

### 9.4.4 Generate Argus Configuration

Run YAIM to generate the Argus configuration for your site: `/opt/glite/yaim/bin/yaim -c -s site-info.def -n ARGUS_server`

At this point, the Argus services (PAP, PDP and PEP Server) must be configured, up and running.



---

## Nagios Probes for Argus

---

A set of Nagios probes for Argus (EMI-2 and EMI-3) are available to monitor the Argus PAP, PDP and PEP Server:

- Argus Nagios Probes Documentation (EMI): <https://twiki.cern.ch/twiki/bin/view/EGEE/ArgusEMINagiosProbes>



---

## **EMIR Publisher for Argus 1.6 (EMI-3)**

---

You can use [EMIR-SERP](#) to publish the Argus resource information to [EMIR](#). EMIR-SERP uses the information already available in the resource BDII and publish it to an EMIR DSR endpoint.

- See the Argus EMIR Configuration to publish the Argus into EMIR:  
<https://twiki.cern.ch/twiki/bin/view/EGEE/ArgusEMIRConfiguration>





---

## Known Issues

---

### 12.1 Timeouts for certificates from CAs that use OCSP

As of late 2013 a few CAs (e.g. the CERN CA) have started using the [Online Certificate Status Protocol](#) (OCSP) in addition to the CRL mechanism to advertise which certificates have been revoked. Some security libraries will then by default contact the OCSP responder of the CA in real time when a certificate from such a CA needs to be validated. If that outgoing traffic happens to be blocked, or if the responder is slow or even unreachable, the operation will eventually time out (see GGUS:105666). Depending on the case at hand, such a timeout need not be a fatal error in itself, but will at least slow down operations and may therefore cause timeouts downstream (e.g. for CREAM or gLExec).

#### 12.1.1 Workaround

In EGI and WLCG we do not need OCSP to work at this time, as the CRL mechanism is still deemed sufficient. Therefore it is reasonable to disable OCSP for the time being and version 1.4.1 of the `argus-pdp-pep-common` rpm does that by default. That version was officially released in [Argus v1.6.3](#) as part of EMI-3 Update 27 (2015-06-10).

### 12.2 Performance issue with Argus PEP Server (EMI-2, EMI-3, all versions)

After a large number of authorization requests, the PDP responses caching mechanism in the PEP Server becomes unstable and the performance of the service deteriorates. It is recommended to completely disable the PDP responses caching mechanism in the Argus PEP Server.

Additionally, the default memory settings for the Argus PEP Server could be too low for production site. It is recommended to allocate at least 1GB memory.

#### 12.2.1 Workaround

##### Disabling the caching mechanism

Completely disabling the PDP responses caching mechanism in the PEP Server configuration solves the performance issue. To disable the cache:

1. edit the `/etc/argus/pepd/pepd.ini` file
2. add the parameter `=maximumCachedResponses = 0=` in the [PDP] section (see example)
3. restart PEP Server: `/etc/init.d/argus-pepd restart` (or increase the memory settings, then restart)

Example:

```
[PDP]
pdps = https://chaos.switch.ch:8152/authz
# disabling the cache
maximumCachedResponses = 0
```

### Increasing the default memory

Allocating 1GB of memory for the Argus PEP Server solves the performance issue. To increase the default memory:

1. edit the `/etc/sysconfig/argus-pepd` file
2. set the line `PEPD_JOPTS="-Xmx1024M"`
3. restart PEP Server: `/etc/init.d/argus-pepd restart`

## 12.3 Problem the EMI-3 update and Argus PEP Server v.1.6.1

Updating the Argus PEP Server to the last EMI-3 update (`argus-pep-server-1.6.1`) will not always restart the Argus PEP Server

### 12.3.1 Workaround

After the update (`yum update`), restart the Argus PEP Server by hand:

```
root# /sbin/service argus-pepd restart
```

## 12.4 Problem with Argus 1.6 (EMI-3) and `fetch-crl`

The Argus metapackage `emi-argus` have no dependency on the `fetch-crl` cron job. Installing and starting the Argus services, without having `fetch-crl` installed, will cause SSL errors when trying to connect to the services.

### 12.4.1 Workaround

**Prior** to installing the `emi-argus` metapackage, install the `fetch-crl` package by hand, and run it at least once:

```
root# yum install fetch-crl
root# fetch-crl -v
```

If you have **already** installed and configured Argus, just install the `fetch-crl` package by hand, run it at least once, and restart the Argus services:

```
root# yum install fetch-crl
root# fetch-crl -v
root# service argus-pap restart
root# service argus-pdp restart
root# service argus-pepd restart
```

## 12.5 Problem with Nagios plugins for Argus and TMP directory permission

The `nagios-plugins-argus` package do not set the correct ownership for the `/var/lib/grid-monitoring/nagios-plugins-argus` temp directory. The plugins will throw errors (Permission denied) because the ownership of the directory is not `nagios`.

### 12.5.1 Workaround

To fix the temp directory ownership to `nagios`, please do:

```
root# yum install nagios-plugins-argus
root# chown -R nagios:nagios /var/lib/grid-monitoring/nagios-plugins-argus
```

## 12.6 Problem with Argus 1.5 (EMI-2) and CREAM

Under heavy load the Argus PEP Server (v1.5.1) does not always return a user mapping for a permitted operation, causing CREAM to throw an error and abort the job. This typically occurs for 10% of the jobs submitted by CREAM.

### 12.6.1 Workaround

Disabling the PDP responses caching mechanism in the PEP Server configuration solve this issue. To disable the cache:

1. edit `/etc/argus/pepd/pepd.ini`
2. add the parameter `=maximumCachedResponses = 0=` in the `[PDP]` section (see example below)
3. restart PEP Server: `/etc/init.d/argus-pepd restart`

Example:

```
[PDP]
pdps = https://chaos.switch.ch:8152/authz
# disabling the cache
maximumCachedResponses = 0
```

## 12.7 Problem with upgrade from Argus 1.4 (EMI-1) to Argus 1.5 (EMI-2)

When upgrading an previous Argus 1.4 (EMI-1) installation on SL5, you need to re-install Argus 1.5 (EMI-2). This is due to an error in the Argus 1.4 post uninstall script.

### 12.7.1 Workaround

Simply reinstalling the components with YUM just after the upgrade solves the issue:

```
yum upgrade
(argus is upgraded...)
yum reinstall argus-pap argus-pdp argus-pep-server
```



---

## Authorization Service: Grid Map File Syntax

---

### 13.1 Description

Grid Map files are used to express a mapping from subject attributes in to a local account.

### 13.2 File Syntax

Each line of a grid map file is either pure whitespace, a comment, or a mapping.

Comments begin with a number or hash sign (#), may be followed with any character and ends at the end of the line. The following is an example of grid map file which contains only a single comment.

```
# This is a map file with a single comment
```

A mapping line consists of two parts, the first is a key that identifies the subject that will be mapped. This part is always enclosed in double quotes ("). The second part is the comma-separated list of targets to which the subject may be mapped. The first and second parts of the map line are separated by any number of spaces or tabs. The following is an example of a grid map file mapping an FQAN to an account.

```
"/vo/atlas/analysis" .atlas
```

### 13.3 Subject Keys

The subject key, the first component of a mapping line, may be either an X.509 DN or a Fully Qualified Attribute Name (FQAN).

If an X.509 DN is used it may be either in the non-standard grid form (where each component is separated by a forward slash (/) and the CN comes last), or the standard RFC2253 format (where components are separated by a comma (,) and the CN comes first). The RFC2253 format is preferred. Both formats may be used within a single file and in both case the following escape sequences may be used:

Escape Sequence	Represents
\'	a single quote
\"	a double quote
\\	a backwards slash
\/	a forward slash
\f	a formfeed
\n	a new line
\r	a carriage return
\t	a horizontal tab
\x##	the ASCII character corresponding to the given hexadecimal digits
\u####	the UTF-8 character corresponding to the given hexadecimal digits

## 13.4 Map Targets

The map target, the second component of a mapping line, is a comma separated list of either an account or group names.

A map target prefixed with a dot . represents a pool account.

## Argus YAIM Configuration for EMI

**Warning:** As of UMD 4.0, YAIM is no longer supported to configure Argus. This page is kept for reference only.

### 14.1 YAIM Configuration for ARGUS\_server

#### 14.1.1 Mandatory General Variables

- `SITE_NAME` BDII site name
- `USERS_CONF`
- `GROUPS_CONF`
- `VOS` List of supported VO names
- `VO_<vo-name>_VOMS_CA_DN` VOMS CA DN for each VO name listed in `VOS`
- `VO_<vo-name>_VOMSES` VOMS definition for each VO name listed in `VOS`

More information on these variables available here: [https://twiki.cern.ch/twiki/bin/view/LCG/Siteinfo\\_configuration\\_variables](https://twiki.cern.ch/twiki/bin/view/LCG/Siteinfo_configuration_variables)

#### 14.1.2 Mandatory Service Specific Variables

They can be found in `/opt/glite/yaim/examples/siteinfo/services/glite-argus_server`

Variable Name	Description	Value type	Version
<code>ARGUS_HOST</code>	Hostname of the Argus node.	FQDN Hostname	1.1.0-1
<code>PAP_ADMIN_DN</code>	User certificate DN of the user that will be the PAP administrator.	Certificate DN	1.0.0-1

#### 14.1.3 Default Service Specific Variables

They can be found in `/opt/glite/yaim/defaults/glite-argus_server(.pre|.post)`

Variable Name	Description
CONFIG_PAP	Set this variable to <code>no</code> if you don't want yaim to create the PAP configuration files
CONFIG_PDP	Set this variable to <code>no</code> if you don't want yaim to create the PDP configuration file
CONFIG_PEP	Set this variable to <code>no</code> if you don't want yaim to create the PEP Server configuration file
PAP_HOME	Home directory of the pap service
PAP_ENTITY_ID	This is a unique identifier for the PAP. It must be a URI (URL or URN) and the same entity
PAP_HOST	Set this variable to another value if PAP_HOST is not installed in the same host as PDP a
PAP_CONF_INI	Configuration file for the pap service
PAP_AUTHZ_INI	Configuration file for the pap service authorization policies
PAP_ADMIN_PROPS	Configuration properties for the pap-admin client
PAP_REPO_LOCATION	Path to the repository directory
PAP_POLL_INTERVAL	The polling interval (in seconds) for retrieving remote policies
PAP_ORDERING	Comma separated list of pap aliases. Example: alias-1, alias-2, ..., alias-n. Defines the or
PAP_CONSISTENCY_CHECK	Forces a consistency check of the repository at startup.
PAP_CONSISTENCY_CHECK_REPAIR	if set to true automatically fixes problems detected by the consistency check (usually mea
PAP_PORT	PAP standalone service port
PAP_SHUTDOWN_PORT	PAP standalone shutdown service port
PAP_SHUTDOWN_COMMAND	PAP standalone shutdown command (password)
PDP_HOME	Home directory of the pdp service
PDP_CONF_INI	Configuration file for the PDP service
PDP_ENTITY_ID	This is a unique identifier for the PEP. It must be a URI (URL or URN) and the same ent
PDP_HOST	Set this variable to another value if PDP_HOST is not installed in the same host as PAP a
PDP_PORT	PDP standalone service port
PDP_ADMIN_PORT	PDP admin service port
PDP_ADMIN_PASSWORD	PDP admin service password for shutdown, reload policy, ..., commands
PDP_RETENTION_INTERVAL	The number of minutes the PDP will retain (cache) a policy retrieved from the PAP. After
PDP_PAP_ENDPOINTS	Space separated list of PAP endpoint URLs for the PDP to use. Endpoints will be tried in
PEP_HOME	Home directory for the pep service
PEP_CONF_INI	Configuration for the pep service
PEP_ENTITY_ID	This is a unique identifier for the PEP. It must be a URI (URL or URN) and the same ent
PEP_HOST	Set this variable to another value if PEP_HOST is not installed in the same host as PAP a
PEP_PORT	PEP service port
PEP_ADMIN_PORT	PEP admin service port
PEP_ADMIN_PASSWORD	PEP admin service password for shutdown, clear cache, ..., commands
PEP_MAX_CACHEDRESP	The maximum number of responses from any PDP that will be cached. Setting this value
PEP_PDP_ENDPOINTS	Space separated list of PDP endpoint URLs for the PEP to use. Endpoints will be tried in

– Main.ValeryTschopp - 11-Mar-2011



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`