Argus Vulnerability Assessment*

Manuel Brugnoli and Elisa Heymann
Universitat Autònoma de Barcelona
June, 2011

Introduction

Argus is the gLite Authorization Service. It is intended to provide consistent authorization decisions for distributed services (e.g. compute elements, portals) in the Grid Environment.

This report presents the results of the vulnerability assessment of Argus [1], version 1.2, as part of a Quality Control process within EMI. For this task we have used the First Principles Vulnerability Assessment (FPVA) methodology [2] proposed by the University of Wisconsin and Universitat Autònoma de Barcelona Middleware Security and Testing Group.

This report is structured as follows. The first section presents the architectural, resource and privilege analysis of Argus. The component evaluation of Argus is discussed in the second section, and the final section provides the results of the vulnerability assessment performed.

Architectural, Resource and Privilege Analysis

In this section, we describe the steps of the FPVA methodology as applied to Argus. We also show the resulting diagrams for each step.

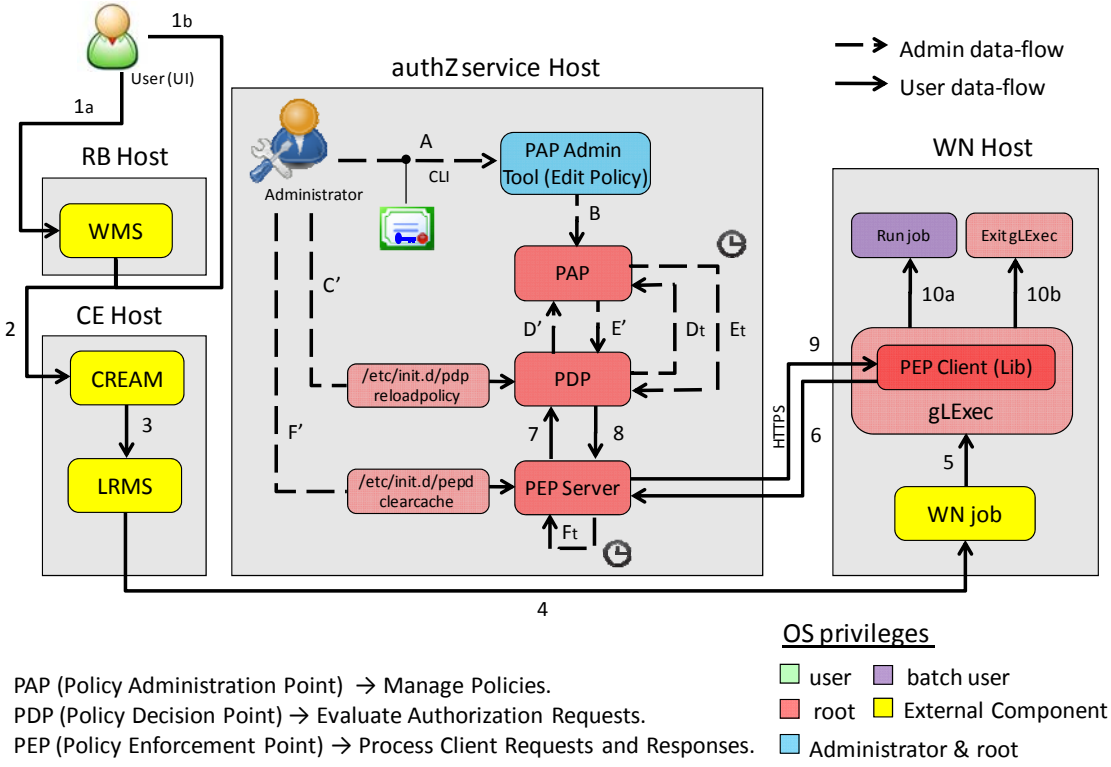
Architectural Analysis

As Figure 1 shows, Argus consists of 3 main components: (a) the Policy Administration Point (PAP) is the component used to create, store and manage the policies used by the Authorization Service; (b) the Policy Decision Point (PDP) to evaluate the authorization requests received from the Policy Enforcement Points against authorization policies retrieved from the PAP; and (c) the Policy Enforcement Point (PEP) to enforce policy decisions.

The PEP component is separated in a client/server architecture, the PEP Server handles the lightweight PEP Client requests, and runs on the Argus host. Lightweight PEP Client libraries are available to authorize requests from the application side, and to enforce decisions locally.

* This research was funded in part by NATO grant ICS.MD.CLG 984138 and the European Commission (contract number: RI-261323), through the EGI-InSPIRE (Integrated Sustainable Pan-European Infrastructure for Researchers in Europe) project.

Argus 1.2 Architecture



PAP (Policy Administration Point) → Manage Policies.
PDP (Policy Decision Point) → Evaluate Authorization Requests.
PEP (Policy Enforcement Point) → Process Client Requests and Responses.

Argus 1.2 Architecture

X' = Optional steps
X_t = Periodic steps

User:

1. *User* submits a job described as a JDL expression.
2. *CREAM* receives a job execution request from *WMS* (1a) or the *User* (1b) directly.
3. *CREAM* sends the job execution request to the *LRMS*.
4. *LRMS* sends the job to the *WN* for its execution.
5. *WN* sends an authorization request to *gLExec*, and *gLExec* interacts with *PEP Server* using an LCMAPS plug-in which uses the *PEP Client* library to check if the mapping request can be satisfied.
6. *PEP Client* sends the request to the *PEP Server*.
7. *PEP Server* sends the authorization request (XACML) to *PDP* for evaluation.
8. *PDP* evaluates the authorization request and sends the response to *PEP Server*.
9. *PEP Server* sends to *PEP Client* the authorization response which can be allowed (10a) or denied (10b).
10. *gLExec* runs job using local identity only if the authorization response is allowed.

Admin:

A. *Administrator* edits policies using the command line interface (CLI).
B. *PAP Admin Tool* writes policies and policy sets and make them available at *PAP*.

C'. *Administrator* forces reload of policies since Argus updates the policies in regular intervals.
D'. *PDP* sends a retrieve policies request to *PAP*.
E'. *PAP* sends policies (XACML) to *PDP*.
F'. *Administrator* sends a clear cache request to *PEP Server* for clearing the response cache.

D_t. *PDP* connects periodically to the remote *PAP* to refresh the repository policy.
E_t. *PAP* sends the policies (XACML) to *PDP*.
F_t. *PEP Server* clears periodically its cache, since *PEP Server* keeps a short response cache.

Figure 1. Argus Architecture Diagram.

The PAP Admin Tool is used to perform all of the Argus policy management operations as well as to set most of the configuration information of the PAP including authorization settings. A valid X.509 certificate or proxy certificate is needed in order to run the PAP Admin Tool.

All interactions between the PAP, PDP, and PEP components can be done over HTTPS. The PEP Server may reuse an existing connection or SSL session when communicating with the PDP in order to minimize the connection overhead.

Furthermore, the PEP Server and the PEP client libraries communicate using the Hessian protocol [3]. It is self-describing (i.e. all information being transmitted is encoded into the messages), and supports binary serialization with many primitives. The Hessian encoded messages are transmitted to the PEP Server by using an HTTP POST, and the body of the POST document is the Base64 encoded.

The use of these secure channels protects sensitive information from being seen or modified by an attacker.

Resource Analysis

The resources used by Argus are shown in Figures 2, 3 and 4. The main resource that Argus manages is the set of policies stored in individual files. All policies are stored in the same directory (*$PAP_HOME/repository*) and the policy files are stored in XACML language.

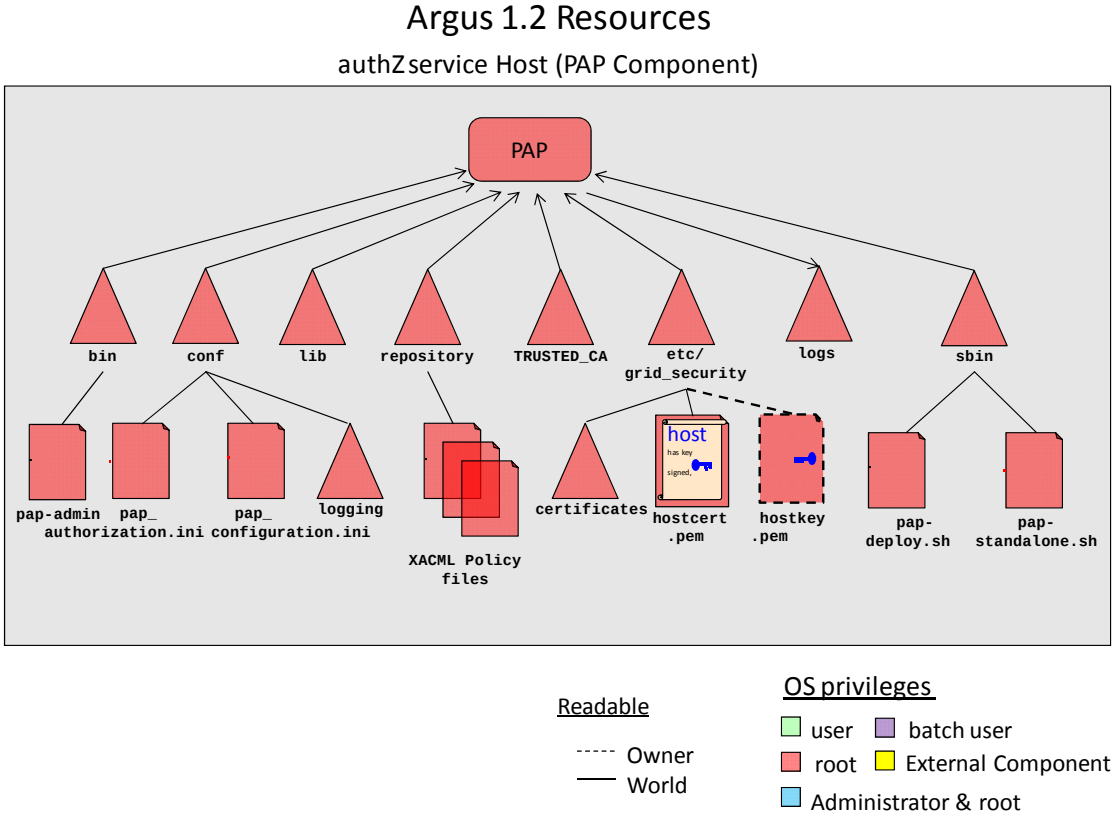


Figure 2. Argus Resources Diagram (PAP Component).

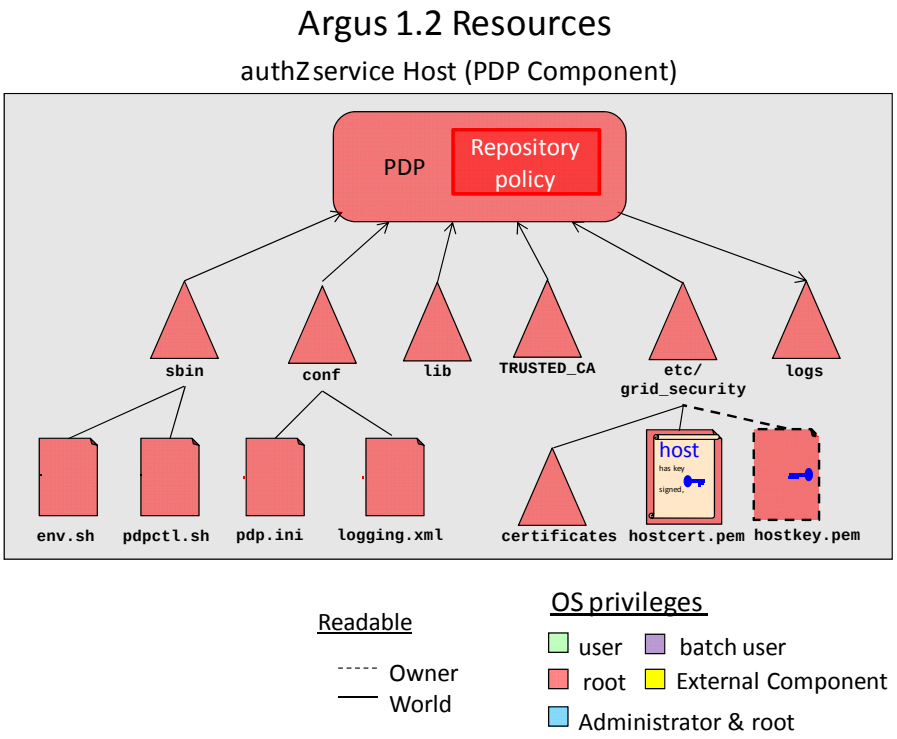


Figure 3. Argus Resources Diagram (PDP Component).

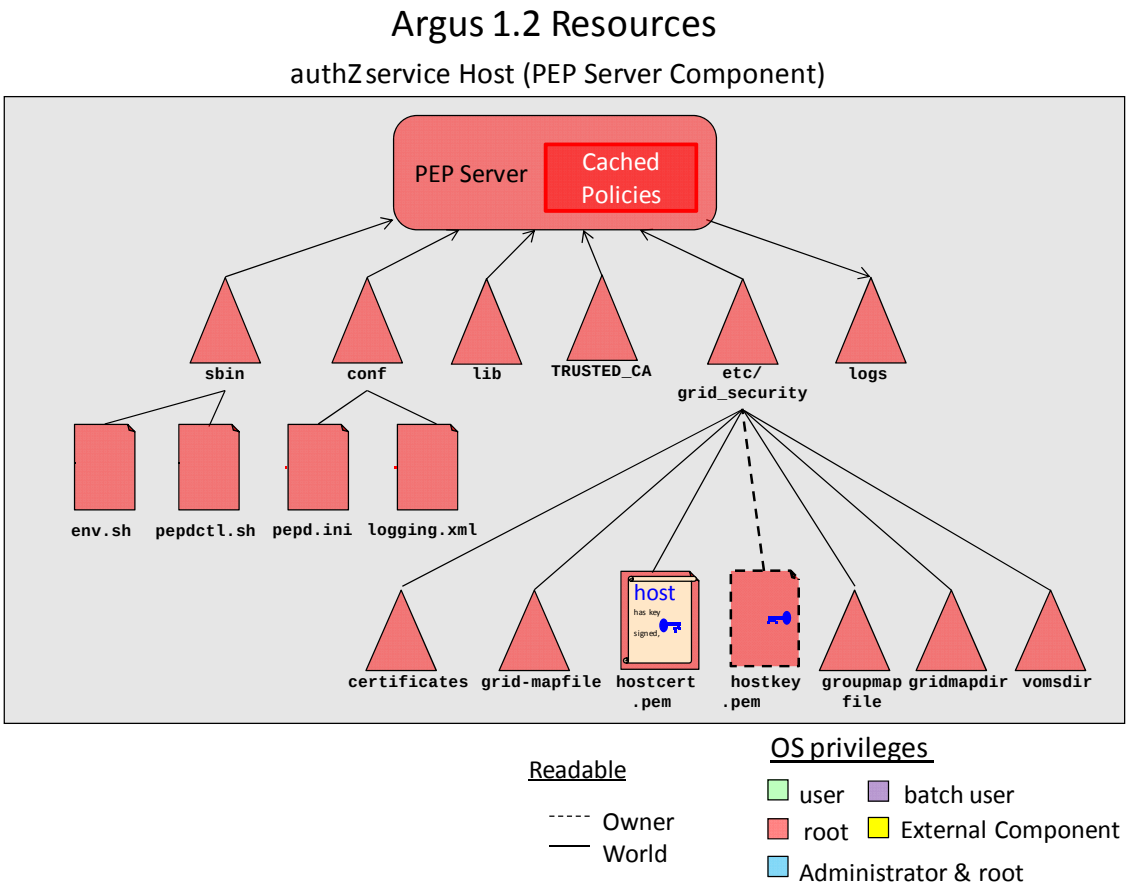


Figure 4. Argus Resources Diagram (PEP Server Component).

The PAP component is configured using two files: *pap_configuration.ini* and *pap_authorization.ini*, both located in the *$PAP_HOME/conf* directory. Most of the information contained in these files can also be set through the PAP Admin Tool (*$PAP_HOME/bin/pap-admin*).

The PDP and PEP components are configured throughout the use of the *pdp.ini* and *pep.ini* files, respectively. These files are a standard INI file and located in the *$PDP_HOME/conf* and *$PEP_HOME/conf* directories.

Argus also uses other files such as the host's credential (public certificate and private key), the set of trusted certificates, some libraries located in the *$COMPONENT_HOME/lib* directory, and finally the log files located in the *$COMPONENT_HOME/logs* are used to keep track of the activity performed by Argus.

Privilege Analysis

The PAP, PDP, and PEP Server run with root privileges in the operating system. The PAP Admin Tool (pap-admin) runs with the same privileges as the user who is executing it. If the user is root, the credentials of the host will be used when connecting to the PAP component, otherwise the user's credential or proxy credential will be used.

Internally the PAP component uses an Access Control List (ACL), composed of several Access Control Entries (ACEs). Each ACE defines the actions that an administrator is allowed to perform. Administrators' privileges are defined in terms of PAP permission flags, whose meaning is the following:

- 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.

The permission flags can be assigned to an administrator by defining an ACE in the *$PAP_HOME/conf/pap_authorization.ini* file or by using the authorization management commands provided by the PAP Admin Tool.

Component Analysis

In this section we explain the different tests carried out during the vulnerability assessment of Argus.

Resource permissions

We checked the permissions of the files that have a high security value and the permissions of these files appeared to be correct. The following table shows the main resources with their permissions:

File Type	Directory	Permissions	Owner	Group
Configuration files	$COMP_HOME/conf/	-rw-r-----	root	root
Libraries	$COMP_HOME/lib/	-rw-r--r--	root	root
Policy repository	$PAP_HOME/repository/	-rw-r--r--	root	root
BIN files	$PAP_HOME/bin/	-rwxr-xr-x	root	root
SBIN files	$COMP_HOME/sbin/	-rwxr-x---	root	root
Log files	$COMP_HOME/logs/	-rw-r--r--	root	root

Client side checks replicated in the server

As explained in the Architectural Analysis, the PAP component has a client called PAP Admin Tool (pap-admin). Using the Argus source code available in public repository, we have modified the *ServiceCLI.java* file with the objective of bypassing the protection mechanisms and generated our own pap.jar file. We also replicated and modified the pap-admin and pap-client-env.sh script files in our user home to change the location of the PAP jar, achieving removing the client-side security checks entirely. However, if an attacker can modify the client-side behaviour to bypass the protection mechanisms, he cannot escalate its access level and perform malicious actions because the security checks are duplicated on the server side code.

Authentication and Authorization Issues

As explained in the Privilege Analysis, the administrators are authenticated and authorized by Argus. If they do not possess a PKI credential signed by a trusted identity provider, or are not authorized to use the PAP, Argus does not permit access to the service. This design makes the system quite strong, facilitates the user management, and dramatically reduces possible threats.

We have tried to run the PAP component using false certificates, but we did not detect any improper verification of X.509 certificates which allow an attacker access without permission. However, we found a bug when an administrator using a valid X.509 certificate (and the corresponding private key) tries to run the PAP Admin tool and this requests the password to decrypt the user's private key. The administrator tries to provide their password, but the PAP Admin does not allow doing that because it asks for the password again. This anomalous behavior happens non-stop. In other words, PAP Admin does not permit to write the password to the administrator by using this authentication method.

We also tried to change the Administrator permission flags to escalate privileges in the PAP Admin tool, but the verification checks seem to be correct.

Network Layer Security

Data sent over a network is both susceptible to eaves-dropping and to modification if precautions are not taken. We studied the architecture diagram in Figure 1 and this showed us that the PEP Server establishes an SSL connection with the PEP Clients to handle the PEP client requests.

This encrypted channel provides strong end-to-end data encryption and integrity. We considered these mechanisms and encryption secure, and we did not perform other tests at this regard.

Injection Attacks

We have checked for the possibility of injection attacks in the PAP Admin command line client in two ways. Firstly by checking the source code to ensure Argus correctly parses and checks the arguments passed through the command line. Secondly by performing injection attacks using special elements[2] through the command line. Appropriate parsing is performed to protect against command injection vulnerabilities and no such vulnerabilities were found.

Variable overflows

We tried to submit large amount of data (up to 3 MB) through the CLI (Command Line Interface). However, we have monitored Argus when carrying out this test and did not detect any dangerous behaviour. Argus also is written in Java, thus does not exist any way to store data into memory that has not been properly allocated.

Revision of previous vulnerabilities documented

As part of our vulnerability assessment of Argus, we checked the security vulnerabilities reported by the Grid Security Vulnerability Group (GSVG) [4]:

- ID #55971: Argus banning by CA does not work.
- ID #59718: Inadequate certificate Validation in Argus.
- ID #56768: Argus may allow a banned user under heavy load.

As result of our evaluation on Argus, we verified that these vulnerabilities were fixed.

Results and Recommendations

After completing the check described in this document we have not found any vulnerabilities in Argus 1.2 at this time, and only a bug in the PAP Admin tool has been reported.

We considered Argus 1.2 is secure enough in terms of architecture and implementation. Some of the different features that made Argus stronger and secure are the following:

- Argus is written in Java. This prevents possible vulnerabilities that we can find in other languages (e.g. C and C++), such as those related to the memory allocation and management.

[2] Sequence of bytes, characters, or words that is used to separate different portions of data within a particular representation or language.

- The fact that a valid X.509 certificate or proxy certificate is needed in order to run the PAP Admin Tool (pap-admin) prevents unauthorized users having access to Argus and facilitates their identification and management.
- The use of a SSL handshake by Argus and their clients prevents exposing sensitive data in the transport layer.
- The design of Argus is solid. The Argus functions are assigned to each component in modular manner so that an attacker cannot access the value information (policies) in unauthorized way or interfere in the authorization process. This design causes that attack surface to be very small.
- The recommended operational configuration of an Argus node is a highly secured host with limited local user access and other services. This reduces the chances of an attack from another account on the machine.

References

[1] Argus Authorization Service, https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework

[2] James A. Kupsch, Barton P. Miller, Eduardo César, and Elisa Heymann. *First Principles Vulnerability Assessment*, 2010 ACM Cloud Computing Security Workshop (CCSW), Chicago, IL, October 2010. http://www.cs.wisc.edu/mist/papers/ccsw12sp-kupsch.pdf

[3] Hessian Binary Web Service Protocol, http://hessian.caucho.com

[4] Grid Security Vulnerability Group – Advisories 2010, http://www.gridpp.ac.uk/gsvg/advisories/