Project no. 032691

KnowARC

Grid-enabled Know-how Sharing Technology Based on ARC Services and Open Standards

Specific Targeted Research Project

Information Society Technologies

D1.6-2 Integration of a Delegation Policy Engine and Policy Parser into KnowARC

Due date of deliverable: June 2, 2008 ***Actual submission date:*** June 9, 2008

Start date of project: June 1, 2006 ***Duration:*** 39 months

Organisation name of lead contractor: NG-UiO

Revision: 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contents

1 INTRODUCTION 1

2 CODE AND DOCUMENTATION 1

3 ARC POLICY PARSING AND EVALUATION INFRASTRUCTURE 2

3.1 ARC SECURITY FRAMEWORK 2
3.2 ARC POLICY EVALUATION ENGINE 5
3.3 CONVENIENT INTERFACE FOR ARC POLICY EVALUATION ENGINE CREATION, ARC POLICY LOADING AND ARC REQUEST CREATION 5
3.4 MATCHING SECURITY ATTRIBUTES TO SPECIFIED POLICY – ARC POLICY DECISION POINT5
3.5 SECURITY ATTRIBUTES COLLECTION AND BASE IMPLEMENTATION 6

4 USING ARC POLICY EVALUATION ENGINE FOR FINE GRAINED IDENTITY DELEGATION 6

4.1 USE CASE FOR FINE-GRAINED IDENTITY DELEGATION 6
4.2 DELEGATION ARCHITECTURE 6
4.3 DELEGATION POLICY INSERTION, EXTRACTION, AND EVALUATION 7

5 CONCLUSION AND FUTURE WORK 7

1 Introduction

This deliverable presents implementation of policy evaluation engine and fine-grained policy constraints mechanism for identity delegation. The current implementation includes ARC policy evaluation engine which can consume the policy and request in ARC specific schema and delegation constraints solution using policy evaluation engine and RFC3820 proxy certificate specifications.

This deliverable consists of this *deliverable report*, the *code* and the related *technical documentation*. It is currently available in 0.9 development branch (hereafter mentioned as ARC1) of ARC middleware and will be available in the production release version 1.0.

In this report we give a short summary of the policy evaluation engine, and the current fine-grained identity delegation solution, as well as some near future work which will serve as the extension of the existing deliverable.

2 Code and Documentation

The code for this deliverable has been developed in the subversion repository: http://svn.nordugrid.org/repos/nordugrid/arc1/trunk/. Basic functionality of policy evaluation engine is located in src/hed/libs/security folder. Its implemention for ARC specific policy along with other pluggable security related components may be found under src/hed/pdc. Some other code performing collection of information used in making authorization decisions resides inside Message Chain Components plugins under src/hed/mcc. Credentials delegation interface and utility are placed into src/hed/libs/delegation and src/clients/credentials correspondingly. The code uses a lot of common functionality of Hosting Environment Daemon infrastructure and can't be easily separated from rest of ARC source tree.

A tarball of the source code is available from the web page of the KnowARC project: http://www.knowarc.eu/download/D1.6-2_code.tgz.

The code is developed using the C++ programming language and is based on a minimal set of external requirements. The following third-party libraries are used:

- *Glibmm* – a part of the gtkmm project[1] – is used for abstraction of the operating system interface. It was chosen to make the future porting to Microsoft Windows easier.
- *Libxml2*[2] provides a way to manage Extensible Markup Language (XML) documents and other structures based on XML like SOAP. A thin layer was added on top of the Libxml2 library to provide a C++ interface to frequently used operations.
- OpenSSL[3] provides the way for creating and parsing proxy certificate, as well as the certificate verification mechanism.
- For building the code, *automake*, *autoconf* and *libtool* utilities are used.

[1] *C++ Interfaces for GTK+ and GNOME*. http://www.gtkmm.org/

[2] *The XML C parser and toolkit of Gnome*. http://xmlsoft.org/

[3] *OpenSSL: The Open Source toolkit for SSL/TLS*. http://www.openssl.org/

- For resolution of external dependencies, the build procedure relies on the *pkg-config* tool.

More information about dependencies may be found in README file.

The core components are documented in two ways:

1. in the form of comments embedded in the code for maintenance and further development and converted into API references by using doxygen utility,
2. in the form of a NorduGrid technical docuemntation[4].

The Technical Manual corresponding to this deliverable is available inside source tree under doc/sec/SecurityFrameworkofARC1.pdf

3 ARC Policy Parsing and Evaluation Infrastructure

In this section, the ARC security framework and policy evaluation engine is summarized.

3.1 ARC Security Framework

ARC1 middleware is based on service container framework called Hosting Environment Daemon (HED) (D1.2-2[5]) which provides a hosting place for various services in application level, as well as a flexible and efficient communication mechanism for services which reside in the hosting environment. Briefly, within this hosting environment incoming and outgoing communication always goes through a chain of message chain components (MCC) and ends in an actual service.

HED contains a framework for implementing and enforcing authentication and authorization. Each message chain component (MCC) or service has a common interface which is in responsible for calling various authentication and authorization functionality. This interface is functional in some plug-ins called SecHandler. Each component (MCC or service) could have two queues of SecHandlers independently for incoming message and outgoing message, both of which will be sequentially executed and return the result.

[4] Aleksandr Konstantinov, Weizhong Qiang, *Security Infrastructure of ARC1*, http://www.knowarc.eu/download/D1.6-2_documentation.pdf

[5] The ARC container (first prototype). https://www.knowarc.eu/documents/Knowarc_D1.2-2_07.pdf

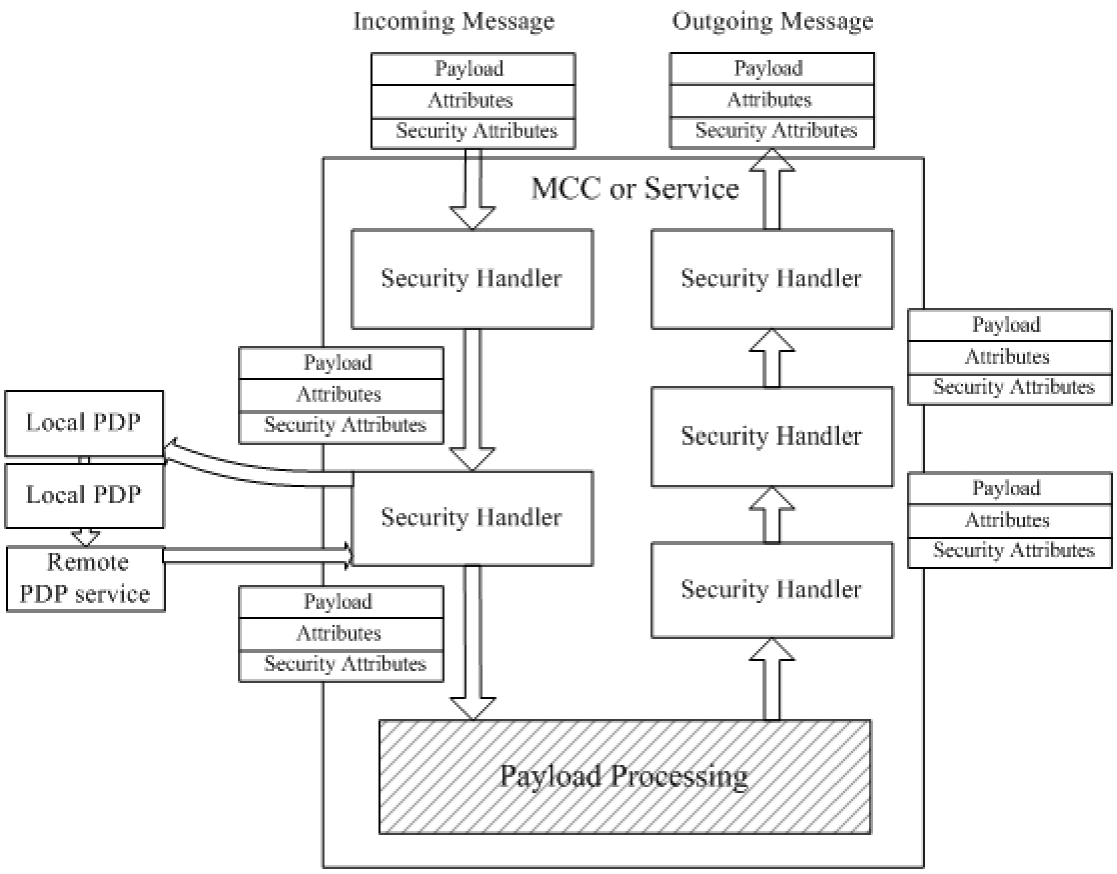

Figure 1. There are two chains of security handler inside the MCC or service. When message comes, each security handler will parse the security attributes which are generated by the upstream MCC/services or probably upstream security handlers in the same MCC/Service, and authenticate or authorize the target entity (together with the incoming/outgoing message) based on the security attributes. The security handler could also change the payload itself, for example, the username token handler will insert the WS-Security Username-Token[6] into header part of SOAP which is the payload of SOAP MCC. Policy decision point (PDP, a logical entity or place on a server that makes admission control and policy decisions in response to a request from a user) is called by security handler and is supposed to make authorization decision. There are two local PDPs and one remote PDP service for demonstration here. In a real deployment scenario, one or any number of PDPs can be configured inside a SecHandler, and those PDPs will be enforced sequentially.

Besides the pluggable characteristic of message chain components (MCC) or service, the SecHandler is pluggable and can be configured by using configuration file as well. Under the SecHandler there could be some pluggable and configurable sub-modules which specifically handle various security functionalities, such as authorization, authentication, etc. The current implemented sub-modules under SecHandler are mostly for authorization, such as ARC PDP which is a policy decision point for ARC specific request and policy schema; but there is a sub-module coming with message level authentication based on WS-Security Username-Token

[6] Web Service Security Username Token Profile 1.1. http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf

profile. *Figure 1* gives the structure of a MCC/Service, and the message sequence inside it.

Table1 shows the configuration of SecHandler for an example service called “Echo” service.

```
  <Service name="echo" id="echo">
<SecHandler name="identity.map" id="map" event="incoming">
    <PDP name="allow.pdp"><LocalName>test</LocalName></PDP>
</SecHandler>
<SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="arc.pdp">
          <PolicyStore>
                <Location type="file">policy.xml</Location>
                <!-- other policy location-->
          </PolicyStore>
    </PDP>
    <PDP name="simplelist.pdp" location="pemittedlist.txt"/>
</SecHandler>
  </Service>
```

Table 1. Echo service includes two SecHandlers, both of which are responsible for authorization. One SecHandler (called identity.map) includes policy decision point (allow.pdp) which will map the identity (distinguished name parsed from X.509 certificate) in the incoming message into one local identity (local linux user name) called “test”. The other (called arc.authz) includes two PDPs: one (called arc.pdp) will compose ARC specific request based on the security attribute parsed from “incoming” message and evaluate against the ARC specific authorization policy which is specified as “policy.xml”; the other (called simplelist.pdp) will compare the identity in the incoming message against an identity list which includes the permitted identities.

3.2 ARC Policy Evaluation Engine

ARC1 defines specific evaluation request and policy schema. Based on the schema definition, one policy evaluation engine is implemented for parsing the policy, and evaluating the request against policy. The design principal of policy evaluation engine is generality by which the implementation of the policy evaluation engine can be easily extended to adopt some other policy schema, such as XACML policy schema[7]. For more detail information about the design of policy engine, as well as the definition about policy schema and request schema, see the *technical documentation*.

3.3 Convenient Interface for ARC Policy Evaluation Engine Creation, ARC Policy Loading and ARC Request Creation

There is some interface which is implemented for conveniently using the policy evaluation engine, including creating the policy evaluation engine object, loading policy into evaluation engine object, creating request object, and evaluating request against policy. All the exposed classes related to policy evaluation engine are designed to be dynamically loadable from shared libraries, which means these objects are created by the class name. The following is a piece of code about creating the policy evaluation engine object. See *technical documentation* for more explanation about the interface.

```
ArcSec::EvaluatorLoader eval_loader;

//Load the Evaluator

ArcSec::Evaluator* eval = NULL;

std::string evaluator = "arc.evaluator";

eval = eval_loader.getEvaluator(evaluator);
```

3.4 Matching Security Attributes to Specified Policy – ARC Policy Decision Point

Security related attributes (such as identity of client, requested action, targeted resource) inside HED and application are collected by different components (MCC and Service).

The ARC policy decision point (ARC PDP, named arc.pdp in code) marshals the security attributes into policy evaluation request (compatible to ARC request schema), calls the policy evaluation engine to evaluate the request against the policy (which is in ARC policy format) repository (policy repository includes all of the policies which have been configured into the PDP), and get back the evaluation result. As *Table1* shows, ARC PDP (together with the policy information inside) can be configured as the sub-node of SecHandlers in the service configuration file.

3.5 Security Attributes Collection and Base Implementation

All of the security attributes are collected from each request and each session and stored together with the message object. In current implementation, there is an interface called Security Attribute which can be implemented by different components (MCC or Service) to collect security attributes. Interface implementation has been provided for all of the MCCs. For instance, TLS MCC implemented the interface to collect all of the identity information about peer certificate, as well as local certificate. Service developer can also implement the interface to collect application level's security attributes.

[7] XACML policy schema. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-policy-schema-os.xsd

Beside the security attribute collecting, the interface Security Attribute can also be used to compose the specific formatted XML object such as ARC specific policy evaluation request. This interface can exactly be used by policy decision point such as ARC PDP to marshal the policy evaluation request.

4 Using ARC Policy Evaluation Engine for Fine Grained Identity Delegation

In this section, the fine-grained delegation solution by utilizing RFC3820 proxy certificate specification to embed the delegation policy and invoking policy evaluation engine to enforce the delegation policy is summarized.

4.1 Use Case for Fine-grained Identity Delegation

The X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile (RFC3820) defines widely used standard for identity delegation and single sign on in Grid community. However current usage practices have some deficiencies.

Commonly only two mechanisms are used for limiting capabilities of generated credentials. Those are limiting depth of delegation and adding time constraints to proxy certificates. Usually no other restrictions are defined, which makes the delegation uncontrollable and unpredictable in a Grid world made of chains of multiple services.

To restrict possible misuse of delegated credentials by untrusted intermediate services some kind of fine-grained delegation mechanism is needed.

4.2 Delegation Architecture

In current implementation delegation is achieved through identity delegation implemented using X509 Proxy Certificates as defined in RFC 3820. Client wishing to allow service to act on its behalf provides Proxy Certificate to the service using Web Service based Delegation interface described in *technical documentation.*

For limiting the scope of delegated credentials along with usually used time constraints it is possible to attach policy document to Proxy Certificate. According to RFC 3820 Policy is stored in ProxyPolicy extension. In order not to introduce new type of object policy is assigned id-ppl-anyLanguage identifier. RFC 3820 allows any octet string associated with such object. We are using textual representation of ARC Policy XML document.

4.3 Delegation Policy Insertion, Extraction, and Evaluation

ARC1 comes with utility “approxy” which generates proxy credentials from certificate/private key pair. ARC Policy XML document can be specified as an argument of this utility to generate ProxyPolicy extension of RFC3820 proxy certificate.

Each deployment which need to enforce fine-grained delegation must use dedicated Security Handler plug-in (named delegation.collector in code) to collect all policy documents from Proxy Certificates used for establishing secure connection.

Then those documents must be processed by dedicated policy decision point plug-in (named delegation.pdp in code) to make a final decision based on collected policies and various information about client’s identity and requested operation. Service or MCC chain supporting Delegation Restrictions must accept negative decision of this PDP as final and do not override it with any other decision based on other policies. Delegation PDP is similar to ARC PDP described above except that it takes its policy documents directly from Security Attributes. Differently from ARC PDP, Delegation PDP is meant to be used for enforcing policies defined by client.

5 Conclusion and Future Work

The deliverable provides a general and extensible policy evaluation engine. Based on the policy evaluation engine, an identity delegation constraining solution is provided for fine-grained delegation.

Some standard policy schema could need to be supported for interoperability. The candidate is XACML which is a widely accepted policy schema.

Currently, for the policy exchanging in delegation scenario, the textual representation of ARC Policy XML document is inserted into ProxyPolicy extension of proxy certificate, and the integrity of policy is guaranteed by the certificate signature. For the policy exchanging in some general scenario, some other solution could be provided to guarantee the integrity of policy, for instance, “SAML profile of XACML 2.0” can be implemented to provide integrity for XACML policy.