NORDUGRID

NORDUGRID-TECH-16
6/8/08

SECURITY FRAMEWORK OF ARC1

W.Qiang, A.Konstantinov

Abstract

This document is about security design concerns and ideas, as well as security framework implementation in the ARC1 middleware.

TABLE OF CONTENTS

1. INTRODUCTION

The security framework of the ARC1 includes two parts of capabilities: security capability embedded in hosting environment, and security capability implemented as plug-ins with well-defined interfaces which can be accessed by hosting environment and applications. The following design concerns were employed when designing:

- Interoperability and standardization

In consistent with the main design concern of the ARC1, interoperability and standardization is considered in security framework. For example, in terms of authentication, PKI infrastructure and proxy certificate (RFC3820 [1]) is used as most of the other grid middle-wares do. Since supporting of standardization is a way for implementing interoperability, some standard specifications have been implemented as prototype and tested, such as SAML specification.

- Modularity and extensibility

Besides the security functionality which is embedded in hosting environment, the other security functionality is implemented as plug-ins which has well-defined interfaces, and is configurable and dynamically loadable. Since the interoperation interface between security plug-in and hosting environment or applications is predefined, it is easy to extend the security functionality in order to support some other security capability by implementing the interface.

- Backward compatibility

The GSI (Grid Security Infrastructure) based mechanism has been a de-facto solution for grid security. The design of security framework should be compatible to it.

2. SECURITY ARCHITECTURE IN HED. SECHANDLER AND PDP

2.1. STRUCTURE OF SECHANDLER AND PDP

In the implementation of the ARC1, there is a Service Container – the Hosting Environment Daemon (HED) (D1.2-2, [2]) which provides a hosting place for various services in application level, as well as a flexible and efficient communication mechanism.

HED contains a framework for implementing and enforcing authentication and authorization. Each Message Chain Component (MCC) or service has a common interface for implementing various authentication and authorization functionality. This functionality is implemented by using pluggable components (plug-ins) called SecHandler. The SecHandler components are C++ classes and provide method for processing messages traveling through Message Chains of the HED. Each MCC or Service usually implement two queues of SecHandlers – one for incoming messages and one for outgoing called “incoming” and “outgoing” respectively. It is possible for MCC or Service to implement other set of queues. Please check documentation of particular component for that particular information. All SecHandler components attached to the queue are executed sequentially. If any of them fails, message processing fails as well.

Each SecHandler is configured inside same configuration file used for configuring whole chain of MCCs. Some of implemented SecHandler components also make use of pluggable and configurable sub-modules which specifically handle various security functionalities, such as authorization, authentication, etc. The currently implemented sub-modules used by some SecHandlers are Policy Decision Point (PDP) components such as Arc PDP which can process ARC specific Request and Policy documents. Figure 1 gives the structure of a MCC/Service, and the message sequence inside it. And Figure 2 shows the configuration of SecHandler components for an example “Echo” service.

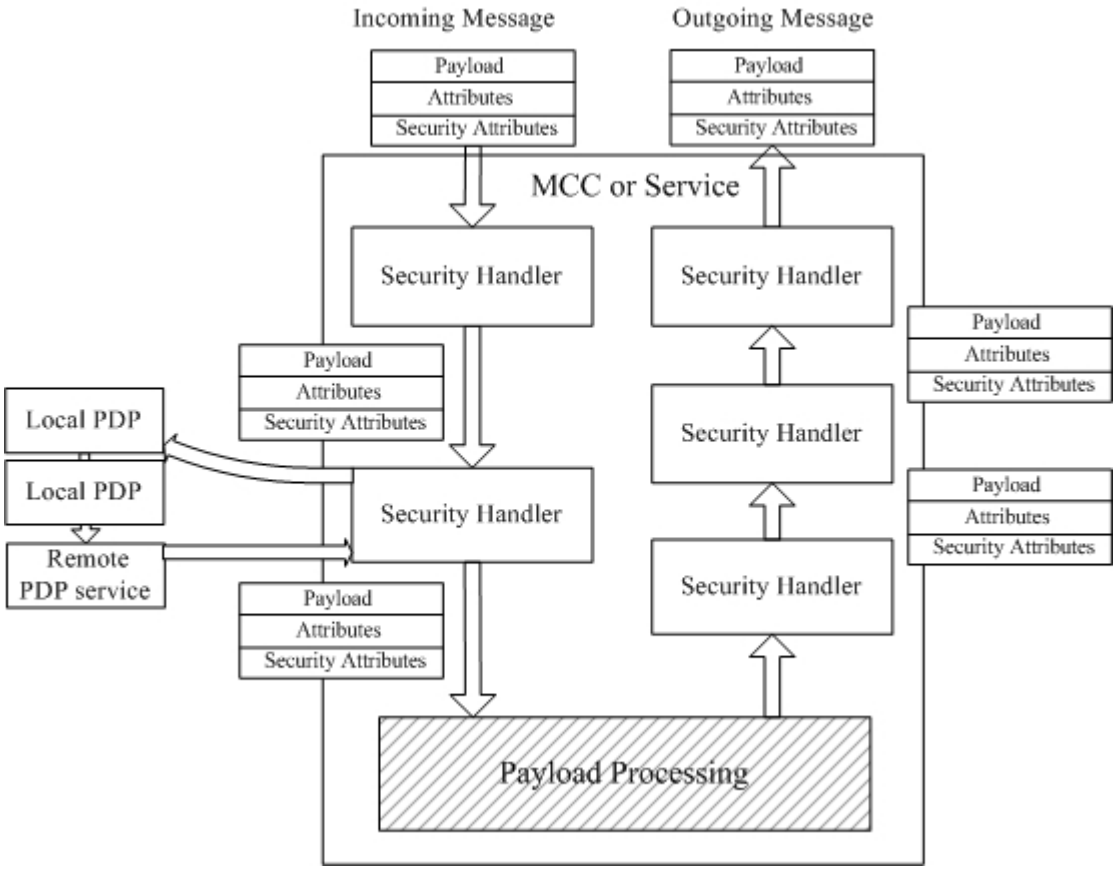

Figure 1. There are usually two chains of SecHandlers inside the MCC or service. Each SecHandler will parse the Security Attributes which are generated by the upstream MCC/services or probably upstream SecHandlers in the same or other MCC/Service, and do message processing or authenticate or authorize the incoming/outgoing message based on the collected information. The SecHandler can also change the payload and attributes of Messsage itself. For example, the Username-Token SecHandler will insert the WSS Username Token [3] into header part of SOAP message. The PDPs are called by the SecHandlers and are supposed to make authorization decision. Here the two local PDP and one remote PDP service is just for demonstration, and one or any number of PDPs can be configured under each SecHandler.

```
<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>
```

Figure 2. Example Echo service is configured to use two SecHandlers, both responsible for authorization. First SecHandler uses the identity of client extracted from the incoming message to map it into local identity like local Linux username. In this case all clients are mapped to local account "test". The second one uses two PDPs: one will compose ARC specific authorization request based on the Security Attributes collected from "incoming" message and evaluate it against the ARC specific authorization policy which is defined in local file "policy.xml"; the other will compare the X509 identity of client extracted from the incoming message against list of identities stored locally.

2.2. INTERFACE OF SECHANDLER

When one component (MCC or service) is loaded according to the configuration information, the SecHandler under the component and the plug-ins like PDP which are attached to the SecHandler will be loaded as well.

There is one simple interface (see Figure 3) defined in class SecHandler, which will be called by the containing MCC/Service once there is message (incoming or outgoing) need to be processed.

```
class SecHandler {
  public:
    SecHandler(Arc::Config*) {};
    virtual ~SecHandler() {};
    virtual bool Handle(Arc::Message *msg) = 0;
};
```

Figure 3. class SecHandler is an abstract class which includes a general interface called Handle which uses Message object as argument. Any security handler implementation should inherit class SecHandler and implement the interface according to the actual functionality. The interface only return simple Boolean value, and any useful information generated during the calling of this interface should be put into the security attribute of the message, or put into the payload itself.

Currently, the ARC1 comes with the following four security handler implemented:

- *arc.authz* – Authorization SecHandler

The *arc.authz* is responsible for calling the interface of policy decision point and getting back the authorization result, and then making decision according to this authorization result. There is one simple interface (see Figure 4) defined in PDP, which will be called by *arc.authz* if configured inside once there is message (incoming or outgoing) need to be processed.

- *identity.map* – Identity Mapping SecHandler

The *identity.map* is a specific authorization oriented security handler. It will map the global identity in the message into local identity like system username based on the result returned by Policy Decision Point components.

- *delegation.collector* – Delegation SecHandler

The *delegation.collector* is responsible for collecting the delegation policy information from the remote proxy credential (proxy certificate is compatible to RFC3820) inside the message, and putting this policy into message's security attribute for the usage of other components, such as *delegation.pdp*.

- *usernametoken.handler* – UseranemToken SecHandler

The task of the *usernametoken.handler* is to generate the WS-Security Username-Token and add it into header of SOAP message which is the payload of outgoing message. It can also extract the WS-Security Username-Token from the header of SOAP message which is the payload of incoming message.

2.3. INTERFACE OF PDP

Figure 4 shows the definition of abstract class PDP. The implementation could be some function which implements the interface by composing the policy evaluation request, evaluating this request against some policy, and returning the evaluation result, or just by composing the policy evaluation request, invoking some remote policy decision web service and getting back the evaluation result.

```
class PDP {
  public:
    PDP(Arc::Config* cfg) { };
    virtual ~PDP() {};
    virtual bool isPermitted(Arc::Message *msg) = 0;
};
```

Figure 4. class PDP is an abstract class which includes a general interface called isPermitted which uses Message object as argument. Any policy decision point implementation should inherit class PDP and implement the interface according to the actual functionality. The interface only return simple Boolean value, and any useful information generated during the calling of this interface should be put into the security attribute of the message, or put into the payload itself.

Currently, the ARC1 comes with the following four policy decision point implementation:

- *arc.pdp* – Arc PDP

The Arc PDP will organize the security attributes into the ARC specific authorization request, call the policy evaluator to evaluate the request against the policy (which is in ARC specific format) repository, and get back the evaluation result. See paragraph 3 for detail information about request schema and policy schema.

- *delegtion.pdp* – Delegation PDP

The Delegation PDP is basically similar to Arc PDP, except it uses the delegation policy parsed from remote proxy credential by *delegation.collector*, and evaluates the request against delegation policy. See section 6. for the design idea and use case of delegation policy in fine-grained identity delegation.

- *simplelist.pdp* – Simplelist PDP

The Simplelist PDP is a simplest implementation of policy decision point. It will match the identity extracted from the remote credential (or proxy credential) with local list of permitted identities.

- *pdpservice.invoker* – PDP Service Invoker

The PDP Service Invoker is a client which can be used to invoke the PDP Service which implements the same functionality as Arc PDP, except that the evaluation request and response are carried by SOAP message. The benefit of implementing PDP Service and PDP Service Invoker is that the policy evaluation engine can be accessed remotely and maintained centrally.

3. POLICY EVALUATION ENGINE

3.1. DESIGN OF POLICY EVALUATION ENGINE

The ARC1 defines specific evaluation request and policy schema. Based on the schema definition, one policy evaluation engine is implemented. 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.

Figure 5 shows the UML class diagram about the policy evaluation engine. It shows all classes and relations simultaneously for getting the overall picture.

The *Evaluator* class is the key class for policy evaluation. It accepts request evaluates it against loaded policy and returns evaluation response.

Three abstract factories - *FnFactory*, *AlgFactory*, *AttributeFactory* - are responsible for creating the *Function*, *CombiningAlg* and *AttributeValue* objects correspondingly. The classes inherited from *CombiningAlg* class take care of implementing various combining algorithms which define relations between <Rule/> elements in policy. The *AttributeValue* type of classes are used for processing different types of <Attribute/> and similar elements. The *Function* classes take care of comparing <Attribute/> elements of request and policy.

The *Policy* class parses <Policy/> or <Rule/> elements and creates *CombingAlg* objects according to

the <RuleCombiningAlg/> attribute of <Policy/>, *Function* objects according to the <Function/> attribute of <Attribute/> and *AttributeValue* objects according to the <Type/> attribute of <Attribute/>. Those objects will be used when evaluating the request.

The *Request* class is responsible for parsing <Request/> element and creates corresponding *AttributeValue* objects according to the <Type/> attribute of <Attribute/>. When evaluating, each *AttributeValue* in request will be evaluated against corresponding *AttributeValue* in the policy by using relevant *Function*.

Due to extensible architecture of code it is relatively easy to add support for new types of *AttributeValue*, *Function and CombingAlg* objects in this way supporting various types of XML based policy languages.

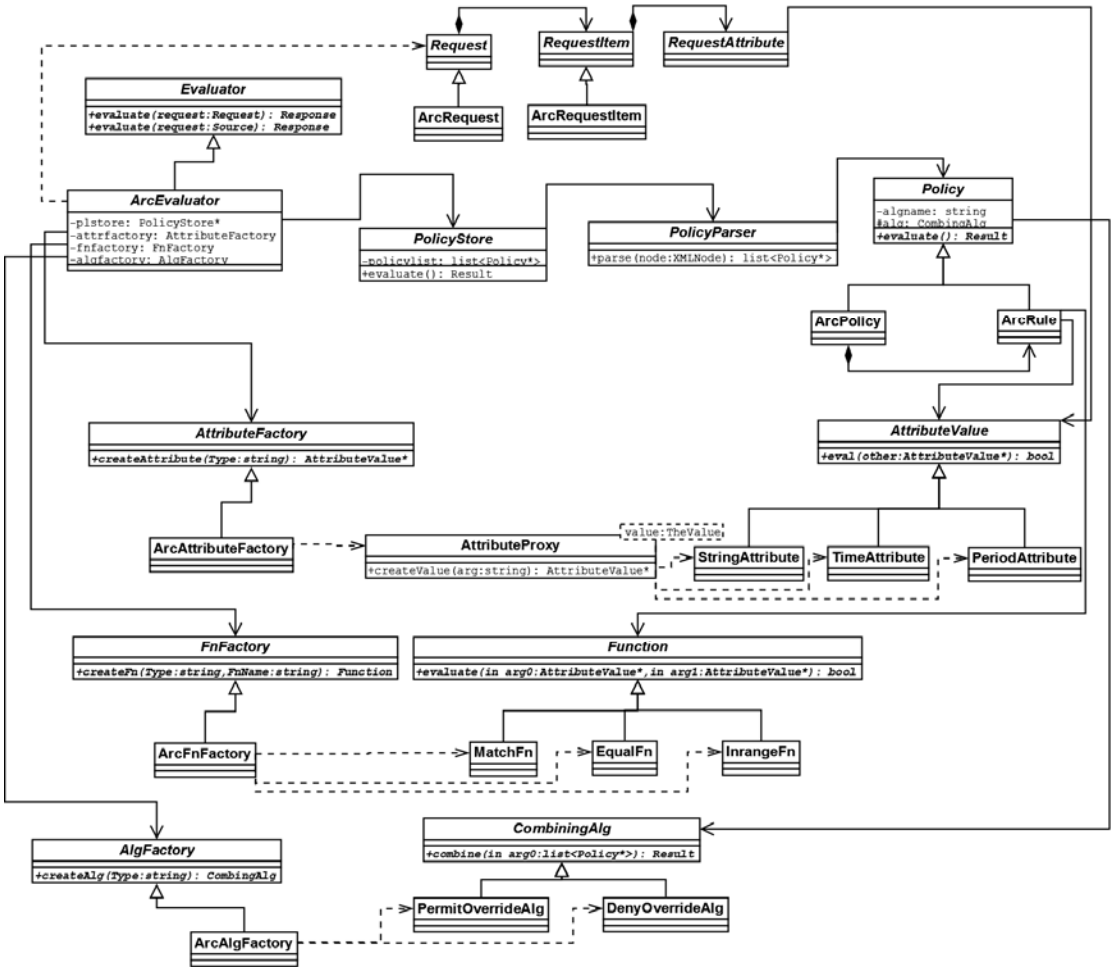

Figure 5. The UML class diagram of the classes inside policy evaluation engine.

3.2. SCHEMAS FOR POLICY EVALUATION ENGINE

The schema for ARC Policy is available at

http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Policy.xsd .

The schema for ARC Request is available at

http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Request.xsd .

The schema for ARC Response is available at

http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Response.xsd .

The ARC Response is not used directly in code. It is in use by PDP Service which provides remote evaluation of policies.

3.3. INTERFACE FOR USING THE POLICY EVALUATION ENGINE

For making usage of policy evaluation engine more convenient basic *Evaluator* class is complemented by additional interfaces. Below are examples of steps needed to carry out policy evaluation and corresponding helper interfaces.

a) Create the policy evaluation object:

```
// Create object which provides an interface
// for loading other objects
ArcSec::EvaluatorLoader eval_loader;
//Load the Evaluator
ArcSec::Evaluator* eval = NULL;
// Define name of policy evaluator.
// This one is for evaluation ARC policies
std::string evaluator = "arc.evaluator";
eval = eval_loader.getEvaluator(evaluator);
```

b) Create the policy object:

```
ArcSec::Policy* policy = NULL;
// Define type of policy – ARC policy in this case
std::string policyclassname = "arc.policy";
// Define source from which policy to be taken
ArcSec::SourceFile policy_source("Policy_Example.xml");
// Load and parse policy
policy = eval_loader.getPolicy(policyclassname, policy_source);
```

c) Create the request:

```
ArcSec::Request* request = NULL;
// Define type of request – ARC request in this case
std::string requestclassname = "arc.request";
// Define source from which request to be taken
ArcSec::SourceFile request_source("Request.xml");
// Load and parse request
request = eval_loader.getRequest(requestclassname, request_source);
```

d) Add the policy into *Evaluator* object:

```
eval->addPolicy(policy);
```

e) Evaluate the request object:

```
ArcSec::Response *resp = NULL;
resp = eval->evaluate(request);
```

The steps d) and e) can also be replaced by:

```
resp = eval->evaluate(request, policy);
```

The *Evalutor::evaluate()* method can also be feed up with both *Policy*/*Request* objects and their sources in any combination. See example code at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/testinterface.cpp for more details about usage of the interface.

The description of mentioned classes and their methods are avaialble in API document at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/doc/KnowARC-API.pdf?format=raw .

4. POLICY DECISION SERVICE

Policy decision service is a service implementation which contains the functionality of ArcPDP. It will accept the soap request containing policy decision request and return soap response containing policy decision response.

The WSDL description of policy decision service is available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/services/pdp/pdp.wsdl . It's configuration is presented in Section 7.5.

5. SECURITY ATTRIBUTES

5.1. INFRASTRUCTURE

Security Attributes represent security related information inside HED framework and store information representing various aspects needed to perform authorization decison - identity of client, requested action, targeted resource, constraint policies.

Each kind of Security Attribute is represented by own class inherited from parent SecAttr class <arc/message/SecAttr.h>. Each Security Attribute stores it's information in internal format and is capable to export it to one of predefined formats using Export() method. Currently only supported format is ARC Policy/Request XML document described in Section 7.1. and 7.2.

Collectors of Security Attributes instantiate corresponding classes and link them to Secuirity Attributes containers - MessageAuth <arc/message/MessageAuth.h> and MessageAuthContext <arc/message/Message.h> storing collected attributes per request and per session correspondingly. Each attribute is assigned a name. Current implementations of Security Attributes Collectors are either integrated into existing MCCs or implemented as separate SecHandler plugins. See Section 5.2. for available Collectors and corresponding Security Attributes.

Note for service developers: Services may implement own authorization algorithms. But they may use Security Atributes as well by providing instances of classes inherited from SecAttr and running them through either configured or hardcoded processors/PDPs.

Processors of Security Attributes are implemented as Policy Decision Point components. Currently there are 2 PDP components available:

- Arc PDP makes use of Security Attributes containing identities of client, resource and requested action. It evaluates either all or selected set of attributes against specified Policy documents thus making it possible to enforce policies defined/selected by service providers.
- Delegation PDP is described below in Section 6.3.

5.2. AVAILABLE COLLECTORS

Here Security Attribute collectors distributed as part of the ARC1 are described except those used for Delegation Restrictions. Those are described in Section 6.2.

5.2.1. TCP

Information is collected inside TCP MCC. The Security Attribute is stored under name 'TCP' and exports ARC Request with following attributes:

Element	***AttributeId***	***Content***
Resource	http://www.nordugrid.org/schemas/policy-arc/types/localendpoint	service_ip[:service_port]
SubjectAttribute	http://www.nordugrid.org/schemas/policy-arc/types/remoteendpoint	client_ip[:client_port]

Table 1. Security Attributes collected at TCP MCC

5.2.2. TLS

Information is collected inside TLS MCC. Generated Security Attribute class is stored under name 'TLS' and exports ARC Request with following attributes:

Element	***AttributeId***	***Content***
SubjectAttribute	http://www.nordugrid.org/schemas/policy-arc/types/tls/ca	signer of first certificate in client's chain
SubjectAttribute	http://www.nordugrid.org/schemas/policy-arc/types/tls/chain	Subject of certificate in client's chain - multiple items
SubjectAttribute	http://www.nordugrid.org/schemas/policy-arc/types/tls/subject	Subject of last certificate in client's chain
SubjectAttribute	http://www.nordugrid.org/schemas/policy-	Subject of last non-proxy

Element	*AttributeId*	*Content*
	arc/types/tls/identity	certificate in client's chain

Table 2. Security Attributes collected at TLS MCC

5.2.3. HTTP

Information is collected inside HTTP MCC. The Security Attribute is stored under name 'HTTP' and exports ARC Request with following attributes:

Element	*AttributeId*	*Content*
Resource	http://www.nordugrid.org/schemas/policy-arc/types/http/path	HTTP path without host and port part
Action	http://www.nordugrid.org/schemas/policy-arc/types/http/method	HTTP method

Table 3. Security Attributes collected at HTTP MCC

5.2.4. SOAP

Information is collected inside SOAP MCC. Security Attribute is stored under name 'SOAP' and exports ARC Request with following attributes:

Element	*AttributeId*	*Content*
Resource	http://www.nordugrid.org/schemas/policy-arc/types/soap/endpoint	To element of WS-Addressing structure
Action	http://www.nordugrid.org/schemas/policy-arc/types/soap/operation	SOAP top level element name without namespace prefix
Context	http://www.nordugrid.org/schemas/policy-arc/types/soap/namespace	Namespace of SOAP top level element

Table 4. Security Attributes collected at SOAP MCC

6. DELEGATION RESTRICTIONS

6.1. 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 it's behalf provides Proxy Certificate to the service using Web Service based Delegation interface described in Section 6.4.

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.

Each deployment implementing Delegation Restrictions must use dedicated Security Handler plugin (see section 5.1.) to collect all Policy documents from Proxy Certificates used for establishing secure connection. Then those documents must be processed by dedicated Policy Decision Point plugin (see section 2.3.) 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.

6.2. DELEGATION COLLECTOR

This Security Attribute is collected by dedicated Security Handler plugin named "delegation.collector" avaialble as part of the ARC1 distribution. It extracts policy document stored inside X509 certificate proxy extension as defined in RFC3820 and described in Section 6.1. All proxy certificates in a chain provided by client are examined and all available policies are extracted.

Extracted content is converted into XML document. Then document is checked to be of ARC Policy kind. If policy is not recognized as ARC Policy procedure fails and that causes failure of communication.

Proxy certificates with id-ppl-inheritAll [5. RFC3820. http://www.faqs.org/rfcs/rfc3820.html] property are passed through and no policy document is generated for them. Proxies with other type of policies including id-ppl-independent are not accepted and generate immediate failure.

6.3. DELEGATION PDP

The Delegation PDP is similar to the Arc PDP described above except that it takes it's Policy documents directly from Security Attributes. Differently from Arc PDP it is meant to be used for enforcing policies defined by client.

6.4. DELEGATION INTERFACE

Delegation interface in the ARC1 is implemented using Web Service approach. Each ARC1 service wishing to accept delegated credentials implements this interface. Here is how delegation procedure works:

- Step 1
    - Client contacts service requesting operation DelegateCredentialsInit. This operation has no arguments.
    - Service responds with DelegateCredentialsInitResponse message with element TokenRequest. That element contains credentials request generated by service in Value. Type of request is defined by attribute Format. Currently only supported format is x509. Along with Value service provides identifier Id which is used in second step.
- Step 2
    - Client requests UpdateCredentials operation with DelegatedToken argument. This element contains Value with serialized delegated credentials and Id which links it to first step. Delegated token element may also contain multiple Reference elements. Reference refers to the object which these credentials should be applied to in a way specific to the service. The DelegatedToken element may also be used for delegating credentials when Step 2 is combined with other operations on service in service specific way.
    - Service responds with empty UpdateCredentialsResponse message.

7. SCHEMAS, DESCRIPTIONS AND EXAMPLES

7.1. AUTHORIZATION POLICY

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Policy.xsd .

7.2. AUTHORIZATION REQUEST

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Request.xsd .

7.3. AUTHORIZATION RESPONSE

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/Response.xsd .

7.4. INTERFACE OF POLICY DECISION SERVICE

WSDL with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/services/pdp/pdp.wsdl .

7.5. CONFIGURATION OF PDP SERVICE

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/services/pdp/pdp.xsd .

Below is an example configuration of PDP service which can evaluate ARC Request against ARC Policy stored in local file.

```
<Service name="pdp.service" id="pdp_service">
<!--The element <Evaluator/>, <Policy/> and <Request/> configuration
      are supposed to be used to load object; element <PolicyStore/> is
      supposed to be used to get the location of policy-->
   <pdp:PDPConfig>
      <pdp:PolicyStore>
         <Location Type="file">Policy_Example.xml</Location>
         <!-- other policy location-->
      </pdp:PolicyStore>
      <pdp:Evaluator name="arc.evaluator" />
      <pdp:Policy name="arc.policy" />
      <pdp:Request name="arc.request" />
   </pdp:PDPConfig>
</Service>
```

See Section 7.7. for the explanation of ARC Policy.

7.6. SIMPLELIST PDP CONFIGURATION AND POLICY EXAMPLE

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/simplelistpdp/SimpleListPDP.xsd .

Below is an example configuration of SimpleList PDP inside “echo” service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="simplelist.pdp" location="simplelist"/>
  </SecHandler>
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

The attribute “name” of <PDP/> is critical for loading the object. Specifically, the name “simplelist.pdp” is for loading the SimpleList PDP object.

The policy file “simplelist” is a local file which contains the list of X509 subjects of authorized entities. It the peer certificate is proxy certificate, the identity in this list should only include the original DN of users's certificate.

For example content of *simplelist* file may look like this:

/C=NO/O=UiO/CN=test1

/C=NO/O=UiO/CN=test2

7.7. ARC PDP CONFIGURATION AND POLICY EXAMPLE

XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/arcpdp/ArcPDP.xsd .

Below is an example of configuration of Arc PDP inside “echo” service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="arc.pdp">
      <PolicyStore>
        <Location type="file">Policy_Example.xml</Location>
        <!--other policy location-->
```

```
    </PolicyStore>
   </PDP>
  </SecHandler>
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

The name “arc.pdp” is for loading the ArcPDP object.

There could be a few policy files under <PolicyStore/>. The request will be checked against all of the policies.

There is an example policy for echo service below. See Section 7.1. for the policy schema. The example policy is made of following elements:

1. Line 14 defines resource being protected. In this it is everything located under HTTP path “/Echo”.

2. Lines 17 and 18 define allowed HTTP operations to be “POST” and “GET”. Line 19 also defines SOAP operation “echo” to be applied to service at path defined above.

3. Lines 10 and 9 require the requester to present X509 certificate with specified identity and signed by specified Certification Authority.

4. No <Conditions/> defined.

5. Line 3 defines that if and only if all of the above constraints have been satisfied by requester, the <Rule/> evaluates to Permit decision.

The Secuirity Attributes used by Arc PDP are collected by different MCCs. It is possible for service to collect some application-specific attributes by implementing class inherited from SecAtt. And that should be the task of application developer.

Administrator of service can configure Authorization SecHandler - arc.authz - for each MCC and Service and define reasonable and meaningful policy. While defining policy the administrator must take into account that the attributes defined in the policy should be already collected by previous components in a chain. For instance, policy with AttributeId “http://www.nordugrid.org/schemas/policy-arc/types/http/path” should not be configured inside SecHandler attached to MCCTLS.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Policy    xmlns="http://www.nordugrid.org/schemas/policy-arc"    PolicyId="sm-
    example:arcpdppolicy" CombiningAlg="Deny-Overrides">
3.    <Rule Effect="Permit">
4.      <Description>
5.        Example policy for echo service
6.      </Description>
7.       <Subjects>
8.         <Subject>
9.           <Attribute      AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/tls/ca" Type="string">/C=NO/ST=Oslo/O=UiO/CN=CA</Attribute>
10.          <Attribute      AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/tls/identity" Type="string">/C=NO/ST=Oslo/O=UiO/CN=test</Attribute>
11.        </Subject>
12.      </Subjects>
13.      <Resources>
14.        <Resource          AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/http/path" Type="string">/Echo</Resource>
15.      </Resources>
16.      <Actions>
17.        <Action           AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/http/method" Type="string">POST</Action>
```

```
18.         <Action            AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/http/method" Type="string">GET</Action>
19.         <Action            AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/soap/operation" Type="string">echo</Action>
20.       </Actions>
21.       <Conditions/>
22.    </Rule>
23. </Policy>
```

7.8. PDP SERVICE INVOKER CONFIGURATION

Configuration XML schema with comments is available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/pdpserviceinvoker/ArcPDPServiceInvoker.xsd .

Below is an example of configuration of PDP Service Invoker inside “echo” service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <!--Remote pdp service invoking-->
      <PDP name="pdpservice.invoker">
        <ServiceEndpoint>https://127.0.0.1:60001/pdp.service</ServiceEndpoint>
        <KeyPath>./key.pem</KeyPath>
        <CertificatePath>./cert.pem</CertificatePath>
        <CACertificatePath>./ca.pem</CACertificatePath>
      </PDP>
  </SecHandler>
  <next id="echo"/>
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

The name “pdpservice.invoker” defines the PDP Service Invoker object.

The PDP Service Invoker is a client of PDP Service. The configuration options include endpoint of service and credentials to be used for establishing secure connection.

7.9. DELEGATION PDP CONFIGURATION

Configuration XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/delegationsh/DelegationPDP.xsd .

Below is an example of configuration of Delegation PDP inside “echo” service.

```
<Service name="echo" id="echo">
  <SecHandler name="arc.authz" id="authz" event="incoming">
    <PDP name="delegation.pdp"/>
  </SecHandler>
  <next id="echo"/>
  <echo:prefix>[ </echo:prefix>
  <echo:suffix> ]</echo:suffix>
</Service>
```

For Delegation PDP, no specific configuration is needed. We only need to switch it on by adding <PDP name="delegation.pdp"/> under <SecHandler/>

7.10. DELEGATION SECHANDLER CONFIGURATION

Below is an example of configuration of Delegation SecHandler inside TLS MCC component.

```
<Component name="tls.service" id="tls"> <next id="http"/>
   <tls:KeyPath>./key.pem</tls:KeyPath>
   <tls:CertificatePath>./cert.pem</tls:CertificatePath>
   <tls:CACertificatePath>./ca.pem</tls:CACertificatePath>
   <!--delegation.collector must be inside tls MCC-->
   <SecHandler name="delegation.collector"
       id="delegation" event="incoming"></SecHandler>
</Component>
```

Current implementation of Delegation SecHandler must be attached to TLS MCC.

7.11. USERNAMETOKEN SECHANDLER CONFIGURATION

Configuration XML schema with comments available at http://svn.nordugrid.org/trac/nordugrid/browser/arc1/trunk/src/hed/pdc/usernametokensh/UsernameTokenSH.xsd .

Below is an example of configuration of UsernameToken SecHandler inside MCCSOAP component.

```
<Component name="soap.service" id="soap">
   <next id="echo"/>
   <SecHandler name="usernametoken.handler" id="usernametoken" event="incoming">
       <Process>extract</Process>
       <PasswordSource>password.txt</PasswordSource>
   </SecHandler>
</Component>
```

UsernameToken SecHandler must be configured under SOAP MCC.

8. USER MANUAL

This section describes how to configure and use SecHandler and PDP elements included in the ARC1 and provides few examples of the ARC Policy documents. The target readers are those users who will use the ARC1 middleware. Currently this section is very short on details. It is going to be continuously extended. Especially taking user feedback into account.

8.1. AUTHORIZATION SECHANDLER AND PDPS

There is a specific Authorization SecHandler (arc.authz) which is implemented for calling the Policy Decision Points (PDP) and serves as their container.

Usually the Authorization SecHandler and included PDPs are used on the service side of communication channel. Although it is also possible to use them on the client side. All possibilities are achieved by modifying the configuration file (hereafter mentioned as service.xml) and possibly providing the authorization policy in a separate file.

Here the “echo” test service is used to explaining the usage, but the explanation applies to other services as well.

The procedure for configuring Authorization SecHandler in service.xml is following:

1. Add the Authorization SecHandler as child element <SecHandler/> of <Service/> element.

   The “name” and “event” attribute of <SecHandler/> element are both important. The “name” attribute is used for distinguishing betwenn loaded SecHandler objects. The “event” attribute defines for which message authorization would be enforced. Usually and reasonably it is done for “incoming” messages. But some services and other Message Chain components may define other internal types of messages. For possible values please refer to documentation of particular Service or MCC. In our particular case “echo” service only supports “incoming” messages for this purpose.

2. Add the PDP configuration as child element <PDP/> under <SecHandler/>. Currently there are four usable PDPs distributed as part of the ARC1 middleware:

- simplelist.pdp – compares Subject of user's X509 certificate to those stored in a file.
- arc.pdp – compares authorization related information parsed from message at various processing steps to Policy document specified in configuration of this PDP.
- pdpservice.invoker - composes the ARC Request, puts request into SOAP message, and invokes the remote PDP service to get the response SOAP which includes authorization decision. The PDP service functionality is similar to arc.pdp.
- delegation.pdp – compares authorization related information parsed from message at various processing steps and Policy document embedded in proxy certificate used by remote side.

Default behavior of Authorization SecHandler is to execute all configured PDPs sequentially till eithe r one of them fails or all produced positive results. This behavior may be modified by attribute “action” of <PDP/> element.

The description of PDP configuration and ARC Policy example are available in Section 7.

8.2. DELEGATION SECHANDLER, DELEGATION PDP AND PROXY CERTIFICATE GENERATION

Delegation SecHandler and Delegation PDP in their current state provide an infrastructure for limiting capabilities of delegated credentials. Their collect and process policies attached to X509 Proxy Certificates respectively. Hence to have delegation restriction working both must be enabled in configuration of service. Configuration of Delegation SecHandler is described in section 7.10. .

The possible location for Delegation PDP is inside Authorization SecHandler (arc.authz). Depending on how fine grained policy of delegated credentials is supposed to be corresponding Authrorization SecHandler may be attached to different MCCs or directly to Service component. However, the precondition for using Delegation PDP is that there must be Delegation SecHandler instantiated earlier in chain.

On the client side, command line utility “approxy” utility can be used to generate Proxy Certificate with Delegation Policy embedded.

Normally approxy appears in $ARC_LOCATION/bin. The usage of approxy is like:

approxy -P proxy.pem -C cert.pem -K key.pem -c constraints

By using argument "-c", some constraints can be specified for proxy certificate. Currently, the life time can be limited by using "-c validityStart=..." and "-c validityEnd=...", "-c validityStart=..." and "-c validityPeriod=...". Like for example

-c validityStart=2008-05-29T10:20:30Z

-c validityEnd=2008-06-29T10:20:30Z

The Delegation Policy can be specified by using "-c proxyPolicyFile=..." or "-c proxyPolicy=...". Like

-c proxyPolicyFile=delegation_policy.xml

The Delegation Policy is the same as the ARC Policy explained in section 7.7. . Simple example below renders delegated credentials usable only for contacting service attached to HTTP communication channel under path /arex (line 6) and allows HTTP operation POST (line 9) on it.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <Policy xmlns="http://www.nordugrid.org/schemas/policy-arc"
3.          PolicyId="sm-example:policy1" CombiningAlg="Deny-Overrides">
4.    <Rule RuleId="rule1" Effect="Permit">
5.      <Resources>
6.        <Resource                                                Type="string"
    AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/http/path">/arex</Resource>
7.      </Resources>
8.      <Actions>
9.        <Action                                                  Type="string"
    AttributeId="http://www.nordugrid.org/schemas/policy-
```

```
        arc/types/http/method">POST</Action>
    10.      </Actions>
    11.   </Rule>
    12. </Policy>
```

Another example of delegation policy is presented below. This policy restricts usage of delegated credentials to SOAP operation CreateActivity (line 5) of Basic Execution Service (BES) [4] namespace (line 9). Such policy could be embedded into credentials delegated to high level Brokering service performing Grid job submission to low level BES on behalf of user.

```
    1.  <?xml version="1.0" encoding="UTF-8"?>
    2.  <Policy    xmlns="http://www.nordugrid.org/schemas/policy-arc"    PolicyId="sm-
        example:policy1" CombiningAlg="Deny-Overrides">
    3.    <Rule RuleId="rule1" Effect="Permit">
    4.       <Actions>
    5.         <Action                                                    Type="string"
        AttributeId="http://www.nordugrid.org/schemas/policy-
        arc/types/soap/operation">CreateActivity</Action>
    6.       </Actions>
    7.       <Conditions>
    8.         <Condition>
    9.           <Attribute                                               Type="string"
        AttributeId="http://www.nordugrid.org/schemas/policy-
        arc/types/soap/namespace">http://schemas.ggf.org/bes/2006/08/bes-
        factory</Attribute>
    10.        </Condition>
    11.      </Conditions>
    12.   </Rule>
    13. </Policy>
```

8.3. USERNAMETOKEN SECHANDLER

The UsernameToken SecHandler is meant for processing - generating and extracting - WS-Security [5] UsernameToken from SOAP header. Hence it must be attached to SOAP MCC of service or/and client communication channel.

On the service side, the functionality of extracting UsernameToken may be configured as described in section 7.11. .

On the client side, the UsernameToken SecHandler may be configured either by using client specific methods (for example see test_clientinterface.cpp src/tests/echo directory of source tree) or through generic client configuration file as shown in example below. This example will generate token with username “user” and password “pass” inside any SOAP message sent by client tools of the ARC1.

```
<ArcConfig>
 <Plugins overlay="add"><Name>arcpdc</Name></Plugins>
 <Chain>
  <Component name="soap.client">
   <SecHandler name="usernametoken.handler" event="incoming" overlay="add">
    <Process>generate</Process>
    <Username>user</Username>
    <Password>pass</Password>
    <PasswordEncoding>digest</PasswordEncoding>
   </SecHandler>
  </Component>
 </Chain>
</ArcConfig>
```

REFERENCES

1. RFC3820- Internet X.509 Public Key Infrastracture (PKI) Proxy, http://rfc.net/rfc3820.html

2. D1.2-2 The ARC container (first prototype), http://www.knowarc.eu/documents/Knowarc_D1.2-2_07.pdf

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

4. OGSA® Basic Execution Service Version 1.0, http://www.ogf.org/documents/GFD.108.pdf

5[⊥] . OASIS Web Services Security (WSS) TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss