NORDUGRID

NORDUGRID-TECH-17
15/11/08

MANUAL OF SECURITY FRAMEWORK

W.Qiang, A.Konstantinov

Abstract

This document is about user manual of security framework, as well as development manual and administration manual.

Table of Contents

1. 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.

1.1. 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

1.2. 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 .

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 . 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>
```

1.3. VOMS Proxy Certificate

The commonly used voms proxy certificate can be used in ARC1 for authentication as normal proxy certificate, and attribute acquiring in order to make authorization decision based on attributes.

1.How to create voms proxy certificate

Currently the proxy creation utility in ARC1—arcproxy can not be used for creating voms proxy certificate because the GSI-based communication requirement from voms server side has not been supported in ARC1 (ARC1 uses standard SSL/TLS communication). So the way to create voms proxy certificate is still to use the “voms-proxy-init” utility.

2.How to use voms proxy certificate in ARC1

The attribute certificate (AC, which is created by voms server and then embedded in voms proxy certificate as one of the certificate extension by “voms-proxy-init” client) will be parsed by TLS handling plugin (called MCCTLS) of ARC1, and saved in the message context in the format “grantor=knowarc://testvoms.knowarc.eu:50000/knowarc:role=guest”.

Service administrator needs to configure the trusted DN chain for verifying the AC under tls MCC's configurationwith the following format. You should specify one node (<tls:VOMSCertTrustDNChain/>) for each vo which you trust. The reason why using trusted DN chain is for service to restrict which vo server is trusted.

In the following table, the first <tls:VOMSCertTrustDNChain> in the following table defines two items: the first one is the DN of voms server, the second line is the DN of the corresponding CA. The second <tls:VOMSCertTrustDNChain> defines regular expression for matching, the DN of voms server and DN of the corresponding CA should both match it. The third one defines the external file which includes the <tls:VOMSCertTrustDNChain>.

```
<tls:VOMSCertTrustDNChain>

 <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/OU=fys.uio.no/CN=ABCDE</tls:VOMSCertTrustDN>

 <tls:VOMSCertTrustDN>/O=Grid/O=NorduGrid/CN=NorduGrid Certification Authority</tls:VOMSCertTrustDN>

</tls:VOMSCertTrustDNChain>

<tls:VOMSCertTrustDNChain>

 <tls:VOMSCertTrustRegex>^/O=Grid/O=NorduGrid</tls:VOMSCertTrustRegex>

</tls:VOMSCertTrustDNChain>

<tls:VOMSCertTrustDNChainsLocation>./pathto/external/file</tls:VOMSCertTrustDNChainsLocation>
```

Service administrator can then use the attribute into access control policy:

```
1.        <Subjects>
2.           <Subject>
3.              <Attribute AttributeId="http://www.nordugrid.org/schemas/policy-arc/
    types/tls/vomsattribute"
    Type="string">grantor=knowarc://testvoms.knowarc.eu:50000/knowarc:role=guest</A
    ttribute>
4.                 <Attribute  AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/tls/ca" Type="string">/C=NO/ST=Oslo/O=UiO/CN=CA</Attribute>
5.                  <Attribute AttributeId="http://www.nordugrid.org/schemas/policy-
    arc/types/tls/identity" Type="string">/C=NO/ST=Oslo/O=UiO/CN=test</Attribute>
6.           </Subject>
7.        </Subjects>
```

1.4. 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 .

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 (also can see share/doc/arc/echo_client.xml.example under the installation directory). 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="outgoing" overlay="add">
    <Process>generate</Process>
    <Username>user</Username>
    <Password>pass</Password>
    <PasswordEncoding>digest</PasswordEncoding>
   </SecHandler>
  </Component>
 </Chain>
</ArcConfig>
```

1.5. X509Token SecHandler

The X509Token SecHandler is meant for processing - generating and extracting - WS-Security [5] X509Token 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 X509Token may be configured as described in section 7.12.

On the client side, the X509Token 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 x509 token (by using the specified certificate and key file) 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="x509token.handler" event="outgoing" overlay="add">
    <Process>generate</Process>
    <KeyPath>/path/to/key.pem</KeyPath>
    <CertificatePath>/path/to/cert.pem</CertificatePath>
   </SecHandler>
  </Component>
 </Chain></ArcConfig>
```

1.6. SP Service and SAML2SSO Service Provider Handler

Service Provider servicer (SP service) implements the functionality of Service Provider in SAML2 Web SSO profile.

Normally SP service will not be deployed independently, instead, it should be deployed together with other services. The SLCS service is the typical deployment about SP service. On the client side, client developer should use the ClientSAML2Interface instead of ClientInterface to call the client functionality. The SLCS client (arcslcs) is the typical usage of ClientSAML2Interface.

Once the SP Service has cooperated with user agent and Identity Provider (external) and succeeded to accomplish the SAML2 SSO profile, the SP service will get the saml authentication assertion which asserts that the authentication has succeeded, and then SP service will store this assertion into session context.

SAML2 assertion consumer handler (saml2ssoassertionconsumer.handler) is the security handler which will understand the authentication assertion and attribute assertion from IdP, and make authorization decision according to the attribute values inside these assertions. Currently the SAML2 assertion consumer handler is an empty security handler and does not effect the services.

The example configuration about using SP service and SAML2SSO assertion consumer handler can be seen in the configuration of SLCS service.

SP service is not only supposed to work together with SLCS service. If it is used to work together with other services, the client should be developed on ClientSAML2Interface instead of ClientInterface, which is the same as that in SLCS client (*arcslcs*).

1.7. SLCS service and client

Short-lived credential service (SLCS) is for signing short-lived x509 credential for user based user's username/password credential. Then the user can use the short-lived x509 credential to access grid services/resources where x509 credential is required by default. SLCS service should depend on the SP (Service provider) service which is one of the participants of SAML2 SSO profile (SAML2 SSO profile is used for authenticating based on username/password credential and getting SAML authentication assertion; and SAML authentication assertion is then used as basis for signing short-lived x509 credential).

a. **SLCS Service**

On the SLCS service side, a typical configuration should be like the following.

```
<?xml version="1.0"?>
<ArcConfig
  xmlns="http://www.nordugrid.org/schemas/ArcConfig/2007"
  xmlns:tcp="http://www.nordugrid.org/schemas/ArcMCCTCP/2007"
```

```
  xmlns:tls="http://www.nordugrid.org/schemas/ArcMCCTLS/2007"
  xmlns:arcpdp="http://www.nordugrid.org/schemas/ArcPDP"
  xmlns:slcs="http://www.nordugrid.org/schemas/ArcConfig/2007/slcs"
>
   <ModuleManager>
        <Path>.libs/</Path>
        <Path>../../hed/mcc/http/.libs/</Path>
        <Path>../../hed/mcc/tls/.libs/</Path>
        <Path>../../hed/mcc/soap/.libs/</Path>
        <Path>../../hed/mcc/tcp/.libs/</Path>
        <Path>../../hed/pdc/.libs/</Path>
        <Path>../../services/saml/.libs/</Path>
        <Path>../../services/slcs/.libs/</Path>
    </ModuleManager>
    <Plugins><Name>mcctcp</Name></Plugins>
    <Plugins><Name>mcctls</Name></Plugins>
    <Plugins><Name>mcchttp</Name></Plugins>
    <Plugins><Name>mccsoap</Name></Plugins>
    <Plugins><Name>arcpdc</Name></Plugins>
    <Plugins><Name>saml2sp</Name></Plugins>
    <Plugins><Name>slcs</Name></Plugins>
    <Chain>
        <Component name="tcp.service" id="tcp">
            <next id="tls"/>
            <tcp:Listen><tcp:Port>60000</tcp:Port></tcp:Listen>
        </Component>
        <Component name="tls.service" id="tls">
            <next id="http"/>
            <tls:KeyPath>./testkey-nopass.pem</tls:KeyPath>
            <tls:CertificatePath>./testcert.pem</tls:CertificatePath>
            <!--tls:CACertificatePath>./cacert.pem</tls:CACertificatePath-->
            <tls:ClientAuthn>false</tls:ClientAuthn>
        </Component>
        <Component name="http.service" id="http">
            <next id="plexer">POST</next>
        </Component>
        <Plexer name="plexer.service" id="plexer">
            <next id="samlsp">/saml2sp</next>
            <next id="soap">/slcs</next>
        </Plexer>
        <Component name="soap.service" id="soap">
            <next id="slcs"/>
              <SecHandler    name="saml2ssoassertionconsumer.handler"    id="saml2ssosp"
event="incoming"/>
         </Component>
        <Service name='saml.sp' id='samlsp'>
            <MetaDataLocation>./test_metadata.xml</MetaDataLocation>
                                  <ServiceProviderName>https://squark.uio.no/shibboleth-
sp</ServiceProviderName>
```

```
        </Service>
        <Service name="slcs.service" id="slcs">
            <next id="slcs"/>
            <slcs:CACertificate>./CAcert.pem</slcs:CACertificate>
            <slcs:CAKey>./CAkey.pem</slcs:CAKey>
            <slcs:CASerial>./CAserial</slcs:CASerial>
        </Service>
    </Chain>
</ArcConfig>
```

SLCS service and SP service should together be configured by using the Plexer. SP service is directly based on http and SLCS is directly based on soap.

SLCS service should specifically include the CA credential (certificate and key file) and the serial number file (which includes the serial number of each signed certificate). Therefore, the service administrator should firstly create a CA credential (note for interoperability purpose in the production grid deployment, the CA credential should be trusted by others).

Note: When deploying the SLCS service, service administrator should deploy a dedicated IdP (Identity Provider) which can be assigned as authentication/attribute authority, or the users should have already had his own IdP. And the IdP information (authentication URL and attribute authority URL) should have already been included into the metadata of above configuration (e.g. test_metadata.xml). On the other hand, the SP (SP service) information (assertion consuming URL) should also have been included into the metadata of the IdP.

After the whole process, user only needs to authentication with his existing IdP and can get back the x509 credential.

Shibboleth IdP (http://shibboleth.internet2.edu) is used for the current solution of IdP, since it is widely deployed for other AAI (Authentication and Authority Infrastructure). There is a test IdP deployed in https://squark.uio.no:8443 (idpname: https://squark.uio.no/idp/shibboleth), you can use the SP from https://sp.testshib.org/ to test the validity of the test IdP. And also you can installed your own IdP which is supposed to authenticates the users from your own organization.

b. **SLCS client**

On the SLCS client side, there is a client utility called “arcslcs”. The command option for arcslcs is as following:

Application Options:

-S, --url=url	URL of SLCS service
-I, --idp=string	IdP name
-U, --user=string	User account to IdP
-P, --password=string	Password for user account to IdP
-Z, --keysize=number	Key size of the private key (512, 1024, 2048)
-K, --keypass=passphrase	Private key passphrase
-L, --lifetime=period	Lifetime of the certificate, start with current time, hour as unit
-D, --storedir=directory	Store directory for key and signed certificate
-c, --conffile=filename	configuration file (default ~/.arc/client.xml)

An example is:

./arcslcs -S https://127.0.0.1:60000/slcs -I https://squark.uio.no/idp/shibboleth -U root -P aa1122 -D /home/wzqiang/arc-0.9/src/clients/credentials -c client.xml

Note user should input the “IdP name” is the corresponding Identity Provider (one of the participants of SAML2 SSO profile) name to which the user would authenticate against by using its username/password credential. And the name is stored inside the metadata on both SP service and IdP provider, and it is used by SP service to get the authentication URL for username/password based authentication.

And the “user” and “password” is the credential which will be used to authentication against IdP.

The “conffile” can be another option for all of the above options, see the following as an example:

```
<ArcConfig>
 <!-- change the paths below to the location of your user certs -->
 <!-- here only trusted certificates are set. Because this arcslcs is
 not supposed to have x509 credential before the samlsso profile has been
 (by using client's username/password credential) passed and the SLCS
 service succeeded to repond this client with the signed x509 credential.
 On the SLCS service's configuration: ClientAuthn should be off.-->
 <!--KeyPath>./testkey-nopass.pem</KeyPath-->
 <!--CertificatePath>./testcert.pem</CertificatePath-->
 <CACertificatesDir>./certificates</CACertificatesDir>
 <SLCSURL>https://127.0.0.1:60000/slcs</SLCSURL>
 <IdPname>https://squark.uio.no/idp/shibboleth</IdPname>
 <Username>root</Username>
 <Password>aa1122</Password>
 <Keysize>1024</Keysize>
 <Keypass>123456</Keypass>
 <CertLifetime>24</CertLifetime>
 <StoreDir>./</StoreDir>
 <Debug>INFO</Debug>
</ArcConfig>
```

There is a temporary Identity Provider (IdP) deployed on *squark.uio.no* for test with the following test username and password: staff, researcher, librarian, binduser; with the same password "123456"

./arcslcs -S https://127.0.0.1:60000/slcs -I https://squark.uio.no/idp/shibboleth -U staff -P 123456 -D /home/wzqiang/arc-0.9/src/clients/credentials -c client.xml

The short-lived credential issued by SLCS service will include the SAML assertion as the extension of X.509 certificate as a proof of passing SAML2 SSO profile.