Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Security Services prototype month 18
D3.5.5

Due date of deliverable: 30/11/2007
Actual submission date: 10/12/2007

Start date of project: June 1^{st} 2006

Type: Deliverable
WP number: 3.5
Task number: 3.5.3

Responsible institution: Rutherford Appleton Laboratory,
Science & Technology Facilities Council,
Harwell Science and Innovation Campus,
Didcot, Oxon OX11 0QX, United Kingdom
Editor & and editor's address: Ian Johnson

Version 1.4 / Last edited by Ian Johnson / 10/12/07

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
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)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.0	18/10/07	Ian Johnson	STFC	first draft
0.1	02/11/07	Ian Johnson	STFC	Expanded on current functionality and limitations/missing features
0.2	05/11/07	Gregor Pipan	XLAB	VOPS instalation guide
0.3	05/11/07	Matej Artač	XLAB	Added the first draft of the VO Policy Service description
0.4	05/11/07	Gregor Pipan	XLAB	Second draft of the VOPS description
0.5	05/11/07	Ian Johnson	STFC	Incorporate comments from team and resolved changes
0.6	06/11/07	Aleš Černivec	XLAB	Added the first draft of the VOPS API
1.0	06/11/07	Ian Johnson	STFC	Identify areas needing more description
1.1	06/12/07	Aleš Černivec	XLAB	Updated user guide for the VOPS
1.2	06/12/07	Ian Johnson	STFC	CDA now only needs PEM files, not JKS format. Incorporate suggestions from XtreemOS partner reviewers
1.3	07/12/07	Ian Johnson	STFC	References include URLs for distributions of CDA and VOPS
1.4	10/12/07	Ian Johnson	STFC	Incorporate suggestions from STFC internal review

Reviewers:
Thilo Kielman (VUA) and David Margery (INRIA)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.5.3	Autonomic Security Policy Management and Enforcement	STFC*, XLAB, ICT

°This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Contents

1 Executive Summary

1.1 Services for Security and VO Management

The first prototype of the services for security and VO management implements a subset of the services specified in D3.5.3, First Specification of Security Services.

The services produced by XtreemOS WP3.5 are the fundamental mechanism to prove a user's identity for authentication purposes in the Grid. These user credentials can then be evaluated in Grid-level VO policies to enable authorization decisions to be made concerning actions requested by subsystems (such as resource matching in AEM job submission) on behalf of the user.

The credentials are contained in an XOS Certificate identifying a user; this is submitted along with, for example, an AEM job request to authenticate the job requester. The user's VO attributes (such as their Global ID) are contained in the certificate and will be used by the VO Policy Service in making VO-level policy decisions, and in mapping to local UIDs/GIDs for node-level access control.

The principle of establishing trust between XtreemOS users and XtreemOS services is to use a Trusted Third Party - the Credential Distribution Authority - to enable clients to trust servers.

The security services use Public Key Infrastructure (PKI) standards and OGF profiles defining public key certificates to reflect best practice and to allow for the possibility of future interoperability with existing Grid middleware authentication systems.

VO policies are expressed in the XACML language, allowing flexibility and the possibility of tool support for creating policies in the future.

The specification of Secure Virtual Organization Management in D3.5.3 defines the following VO services:

- *VO Management Service* to allow creation of VOs, management of VOs, and checking of VO membership
- *VO Policy Service* A Policy Decision Engine for VO-level policies
- *Credential Distribution Authority* (CDA) to issue XOS Certificates to XtreemOS users
- *Resource Certification Authority* (RCA) to issue resource certificates to hosts
- *Identity Service* and *Attribute Service* are supporting services only used by the VO Management Service, CDA and RCA.

1.2 Current Limitations and Missing Functionality

Of the services listed above, the Resource Certification Authority is not planned for delivery in the first prototype. Work on the RCA will start in M19.

The Identity and Attribute services are internal services, not directly visible to users of the security services.

The functionality of the VO Management service is represented in the current version of the first prototype by a hard-coded VO setup, sufficient for authenticating and issuing an XOS Certificate for a pre-defined set of three users.

1.3 Standards and Profiles used

The fundamental principles and standards for Public Key Infrastructure are described in IETF RFC3280 [7]. The XOS Certificate conforms to the Public Key Certificate profile defined in RFC3280, and also to the Proxy Certificate Profile defined in RFC3820[8]. The XOS Certificate conforms to the most relevant recommendations in the OGF Grid Certificate Profile [6].

The VO Policy Service defines policies in the XACML 1.1 language [5]. The CDA, VOPS and RCA are first defined in [3], First Specification of Security Services, and are further refined in [4], Second Specification of Security Services.

2 Prototype Description

2.1 Functionality in the first prototype

We define an XtreemOS user as a user that has registered with a VO and been accepted as a member of that VO - this process normally involves some vetting of registrations by an offline process (e.g. a VO administrator calling the applicant to check their identity).

The first prototype provides a means for an XtreemOS user to request an XOS Certificate, proving their identity within the VO and carrying with it the users' attributes, such as their Global ID in the VO and the VO groups that they belong to. Policies at the VO level are dealt with by the VO Service.

The VO Management Service provides the following functions -

- *VO Membership* - access to a database of VO members
- *Credential Distribution Authority* - providing XOS Certificates to XtreemOS users

The VO Policy Service provides the following functions -

- *Administration* - loading and deleting policies
- *Information* - listing policies
- *Decision Engine* - taking requests and making decisions

2.2 Architecture of the first prototype

To provide an adequate level of trust in the credentials issued by the CDA, we propose a small number of highly trusted servers running the CDA service - initially, we propose just one CDA service for use during the creation of the XtreemOS integrated prototype.

Requests to the CDA service can be made from a CDA client program, which can run on an XtreemOS client machine.

3 Individual Services in the First Prototype

This section describes in more detail the functionality provided by the susbsystems comprising the VO Management Service, and the VO Policy Service. Currently, the VO Management Service currently comprises the VO Membership Service and the Credential Distribution Authority.

3.1 The VO Membership Service

The main functionality of the VO Membership service is to provide a database of VO users. In the current version of the first prototype, a static VO configuration is hard-wired into the CDA service for the purpose of testing the CDA service functionality. No VO Administration functionality is available yet.

The VO Membership service provides means of authenticating a user in a VO. In the current version, this is a check of username and password against details stored in an authentication database. Future versions of the membership service will contact the appropriate Home Authentication Authority to authenticate users.

3.2 The Credential Distribution Authority

An XtreemOS user requests an XOS Certificate by running the CDA client to contact the CDA service. The user is prompted to specify the name of the VO, and their username and password in the VO. The CDA client generates a keypair (consisting of a private key and its corresponding public key). The private key is stored in an encrypted form, protected by a user-specified password.

The CDA client and server use the SSL protocol to establish a secure communications channel before transmitting any information. The client performs server authentication by checking that the CDA server certificate presented in the SSL handshake contains the same Subject principal as a local copy of the CDA public certificate accessible by the CDA client.

The CDA client authenticates the user against the specified VO by sending the VO name, VO username and password to the CDA service, which calls the VO Membership Service. If this authentication succeeds, the CDA client sends the user's public key to the CDA service.

Upon receiving the user's public key, the CDA service generates an X509v3 public key certificate in the format given in the appendix of D3.5.3. The user's VO attributes are added as certificate extensions, and the XOS certificate is returned to the CDA client program. This program verifies the certificate, checking that it has been issued by the trusted CDA service (by checking the Issuer principal in the XOS Certificate against the Subject principal in a local copy of the CDA public certificate).

3.3 VO attributes

The attributes carried in the XOS Certificate fall into the following categories:

Attributes for XtreemFS to facilitate file sharing on a global scale:

- *GlobalPrimaryVOName* - The primary VO that this user is a member of
- *GlobalSecondaryVONames* - A list (possibly empty) of secondary VOs that this user is a member of
- *GlobalUserID* - The globally unique user name for this user, comprising the unique VO name plus the user name
- *GlobalPrimaryGroupName* - The globally unique name of this user's primary group
- *GlobalSecondaryGroupNames* - A list (possibly empty) of secondary groups that this user is a member of

Attributes for generating local UIDs/GIDs and for VO Policy Service and local policy decision points:

- *Role* - The role of the user
- *Group* - The group this user belongs to
- *Subgroup* - The subgroup containing this user

The eight attributes listed above are well-defined and can be processed with the current version of the first prototype.

3.4 Extracting user VO attributes from the XOS Certificate

The VO attributes in a XOS Certificate can be retrieved by the method getVOAttributeValue(X509Certificate cert, VOAttribute attr) in the CertificateProcessor class. Given an enum for the required VO Attribute, this returns the String value of tthe attribute.

Field name	Type	Explanation
host	java.net.InetAddress	The address the node's daemon is running at.
gateway	java.net.InetAddress	The address of a gateway to the subnet with the node.
port	int	The port the node's daemon is listening for traffic at.

Table 1: : CommunicationAddress class fields.

3.5 Description of the VO Policy service

3.5.1 Background information

Selection of the nodes available for job processing follows the data flow depicted by Figure 1. The XJobMng service receives the job's resource requirements in an XML formatted according to the JSDL schema. The service uses the JSDL as a resource query for the XResMng service which returns a collection of addresses pointing to the nodes that conform to the resource query. The VOPS service checks whether the VO policies grant the user, identified by an XOSD certificate, the access to the resources. The collection of permissible nodes returns to the XJobMng service in a signed and certified form.

Resource query The JSDL schema for XML files represents a job description, packed along with the resource requirements for the job. JSDLs can therefore be used for querying for computation nodes that can provide the resources required by the job.

Figure 2 shows a sample JSDL file. The example represents a simple job that runs latex. The JobDefinition.JobDescription.Resources section expresses the resource requirements as required RAM, CPU speed, ...

3.5.2 Node address

The node address structure represents the network address of a node and the port the node's daemon listens for the requests at. Table 1 shows the details of the CommunicationAddress class that holds the node address.

3.5.3 VO Policy Description

The policy structure follows the XACML schema for XML documents. Essentially, the descriptor contains a set of subjects that require certain resources or actions, a set of resources subject to policies, and a list of rules that describe the actual policies that should be enforced when subjects try to access the resources. Figure 3 shows an example policy structure which defines a set of subjects to be the users whose accounts reside at users.example.com. The resources subject to

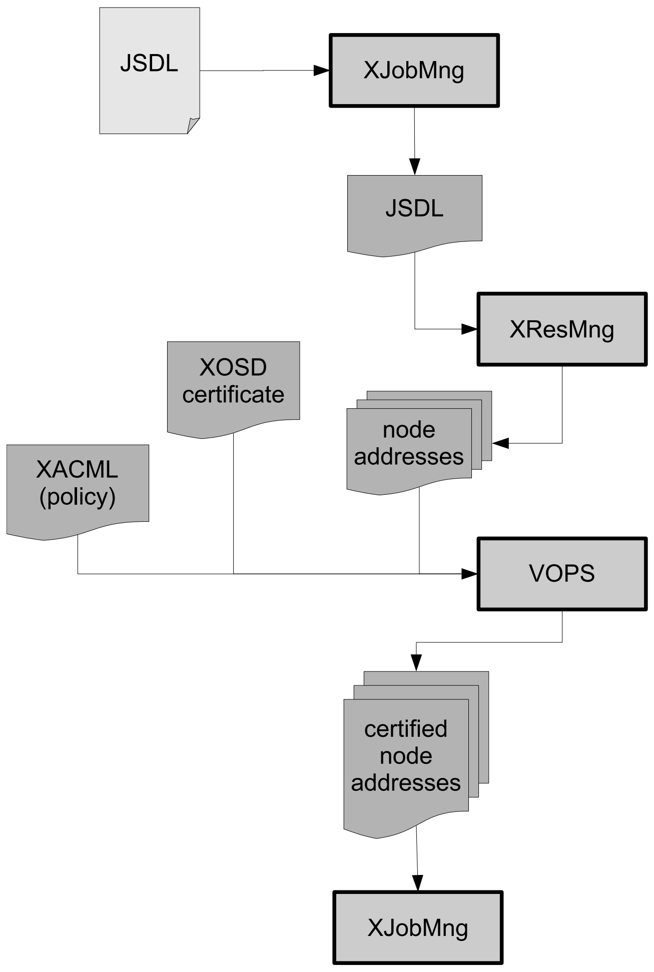

Figure 1: Node selection data flow.

the policies have a common address server.example.com. The action permissible by the subjects is performing the CVS commit.

To test the policy enforcement, another XML using the XACML schema can be used. The structure shown at Figure 4 shows a structure for submitting a commit action by the user seth that is a member of developers group to the server.example.com.

```
<?xml version="1.0" encoding="UTF-8"?>
<JobDefinition xmlns="http://www.example.org/"
  xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
  xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <JobDescription>
    <JobIdentification>
      <JobName>My sample job</JobName>
      <Description> A sample JSDL.</Description>
    </JobIdentification>
    <Application>
      <ApplicationName>latex</ApplicationName>
      <POSIXApplication>
        <Executable>/usr/local/bin/latex</Executable>
        <Argument>/home/user/thesis.tex</Argument>
        <Input>thesis.tex</Input>
        <Output>thesis.div</Output>
      </POSIXApplication>
    </Application>
    <Resources>
      <IndividualPhysicalMemory>
        <Range>
          <LowerBound>268435456.0</LowerBound>
          <UpperBound>1073741824.0</UpperBound>
        </Range>
      </IndividualPhysicalMemory>
      <IndividualCPUCount>
        <Exact>1.0</Exact>
      </IndividualCPUCount>
      <IndividualCPUSpeed>
        <LowerBoundedRange exclusive="true">
                1073741824.0
        </LowerBoundedRange>
      </IndividualCPUSpeed>
      <CPUArchitecture>
        <CPUArchitectureName>x86</CPUArchitectureName>
      </CPUArchitecture>
      <OperatingSystem>
        <OperatingSystemType>
          <OperatingSystemName>LINUX</OperatingSystemName>
        </OperatingSystemType>
      </OperatingSystem>
    </Resources>
  </JobDescription>
</JobDefinition>
```

Figure 2: A sample JSDL.

4 Installation and Configuration of Security Services

4.1 The VO Management Service

The main functionality in the VO Management Service at present is the Credential Distribution Authority. This requires the Java Runtime Environment version 1.6 or later to run.

4.1.1 Requirements

The Credential Distribution Authority is a Java application and needs JRE 1.6 or later to run.

The following JAR files must be on the CLASSPATH:

```
bcprov-jdk16-137.jar
mail.jar
```

The first is the Bouncy Castle library providing a Java Cryptography Extension (JCE) provider. Version 1.37 or later is required. This library is used for creating and processing public/private keypairs and public key certificates.

The second is the JavaMail library, and provides a definition of a Java class for representing email addresses. The CDA doesn't send or process mail messages.

4.1.2 Installation

The CDA prototype is released as a tarball [1]. When the distribution archive is untarred. the following files will be extracted:

```
CDA.jar
README
(directory) keys
-- cdacert.pem
-- cdakey.pem
(directory) lib
-- bcprov-jdk16.137.jar
-- mail.jar
runclient
runserver
```

The classes for the CDA client and CDA service are contained in the CDA.jar file. The 'lib' subdirectory contains the two required JAR files described above. The shell files *runclient* and *runserver* are used to run the client and server respectively.

Files used by CDA Service The CDA service needs to read its private key and the CDA server certificate - these can be stored in the working directory of the CDA service. The private key for the test CDA can be stored in unencrypted form (purely for the purpose of testing), or encrypted.

The CDA certificate is used to obtain the Issuer Principal used in the XOS Certificate. The CDA private key is used to sign the XOS Certificate, proving that is was issued by the entity whose public key is stored in the CDA certificate.

- *cdakey.pem* Private key of the CDA service
- *cdacert.pem* Public certificate of the CDA service

4.2 Running the CDA Service

Use the supplied script 'runserver', specifying a TCP port number that the server should listen to, for example:

```
./runserver 9090 &
```

This port number, and the name of the host running the CDA service, have to be advertised to XtreemOS users wishing to access the CDA.

Files used by CDA Client The CDA client uses the public certificate of the CDA server to verify the identity of the issuer of the XOS Certificate.

The CDA client also uses the CDA service public certificate in negotiating the SSL protocol handshake with the CDA server.

- *cdacert.pem* Used by client to verify the source of the XOS Certificate it has received

Files generated by CDA Client If the user authenticates against the CDA service successfully, the following files are generated:

- *Private key* User private key, encrypted by passphrase (default filename is $privkey.pem$)
- *User Certificate* XOS Certificate holding the public key associated with the user's private key and containing user VO attributes (default filename is $usercert.pem$)

4.3 Installation and Configuration of the VO Policy Service

The VO Policy Service - VOPS is based on the infrastructure developed by WP3.3, which provides following common facilities wrapped in a simple to use framework. The facilities are: Event driven communication layer, providing TCP and SSL capabilities, configurable service infrastructure based on the concepts of the Staged Event Driven Architecture [9], configuration file framework, automatic service wrapper generation and a set of supporting services like XML parser, Cron service, and more.

In the following sections we shall first describe required software to start VOPS service, then we shall describe the installaton procedure, and finally, we shall give some basic information about the configuration file used by VOPS service.

4.3.1 Requirements

VOPS is developed in Java and therefore needs JRE 1.6.* for execution. In addition to standard java libraries following jar files have to be included in the $CLASSPATH$ environment variable:

```
vsuite-findbugs-0.4b.jar
slf4j-api-1.4.3.jar
mina-core-1.1.2.jar
mina-filter-ssl-1.1.3.jar
log4j-1.2.14.jar
commons-logging-1.1.jar
dom4j-1.6.1.jar
```

4.3.2 Installation

As the final packages will be provided by WP 4.1, the installation procedure provided in this workpackage is not ment to be final, and is to be used only until packages will be provided in the M24. The installation is provided as a tarball containing all the files [2], which are needed to run the VOPS service. To unpackage the service, user has to execute following steps:

```
cd ~/my/vops/directory
tar xfvz aem-vops-rev1168-clean-compiled-source.tar.gz
rm aem-vops-rev1168-clean-compiled-source.tar.gz
```

The tarball file contains the initial configuration file - $XOSdConfig.conf$, which can be used as a template for the configuration. The details about how to configure the service are provided in the following subsection.

Certificate file The procedure to create a server certificate involves the 'openssl' command and will be described fully in a later version of this document.

4.3.3 Configuration

VOPS v1.0, which is developed for the M18 is configured to run as a one instance service per VO. As such the service has to make itself known to other services using its functionality.

Configuration file - vops.conf is looked for in following directories in the order in which the directories are listed:

```
~/my/vops/directory/AEM_VOPS_clean/Support/XOSdConfig.conf
./ - working directory
```

The configuration file the following has to provide VOPS the access points to the services, to which it will register its availability. At this moment VOPS registers to two services AEM and XtreemFS. The configuration file in this section looks like following:

```
AEM.AccessPoint = 21.2.132.24:60000
XFS.AccessPoint = 21.2.132.24:10025
```

VOPS provides only an SSL socket connection and has therefore to load a site certificate, in order to process an SSL handshake. certificate can be provided in the same directory as the configuration file, and has to be named *vops.cert*. In addition the configuration file can define a new certificate file name.

```
Certificate.filename = ~/my/vops/directory/AEM_VOPS_clean/
Support/cert/vops.cert
```

The VOPS has to load VO policies, which will define resource usage. The policies are by default stored in the *./policy* directory, but an override can be defined:

```
Policy.directory = ~/my/vops/directory/AEM_VOPS_clean/
VOPS/files/policy
```

VOPS will load all the policies defined in this directory at the start of the service. Because the VOPS has been written as a service of the AEM (build uppon AEM), user has to use XConsole application provided by the AEM in term to use the functionalities provided by the VOPS.

5 User Guide for Security Services

5.1 Using the CDA Client to obtain an XOS Certificate

The supplied shell script 'runclient' takes the following arguments:

- *host* The DNS entry or IP address of the machine running the CDA service
- *port* The port that the CDA service is running on
- *keyfile* Optional filename to store the user's private key (default: privkey.pem)
- *usercert* Optional filename to store the user's XOS Certificate (default: usercert.pem)

An example of running the CDA client to connect to the CDA service:

```
./runclient localhost 9090 mykey.pem mycert.pem
```

The user is prompted for the following information:

- *voname* VO that the user is registered with (currently, only 'esvo' is configured)
- *username* User's name in this VO (currently, users are 'alice', 'bob', and 'eve')
- *password* Password for this VO user (currently, passwords are 'alicepassword', 'password' and 'evepassword' respectively)

If the user supplies the correct details and is successfully authenticated, they are then prompted for the following item:

- *passphrase* A passphrase to protect the user's private key (8 characters or more, must be entered twice to confirm)

The user's private key is written to the specified file ($mykey.pem$ in the example), and the user's XOS Certificate requested from the CDA service and stored in the specified file ($mycert.pem$ in the example).

5.2 User Guide for VO Policy Service

VOPS functionality is divided into two parts. One part is primarely called by other services in the XtreemOS like JobMng, which require policy decision. This part is also called Policy Decision Point or PDP, and is described in the specification deliverables provided by this WP. The second part provides administration interface to add, remove and change VO policies. This functionality provides an Policy Information Point - PIP, also described in previosly mentioned documents.

The simplicity of the first prototype is obvious but to exploit the strength of the XACML policy language, a framework has to be written. First VOPS prototype enables user to create, remove and modify policies as described below.

Policy is composed of two main sections:

- TARGET - which resource does this policy apply to,
- RULEs - set of rules which enforce limited actions that can be taken with resource, described in TARGET section.

When user (he will have to present his identity through certificate, maybe it would be correctly to use the word *administrator*) wants to create policy for some resource usage, he must first create a policy. When creating a policy, he must provide policy's id, policy's description (for now this has to be one string without spaces, reason for this is limitation when using XConsole) and resource, to which this policy will apply to. Using XConsole, user creates policy with

```
xcreatePolicy -pid HomeComputer2 -desc
ThisIsForTestingPurposes -res 192.168.2.102
```

This creates policy for resource *192.168.2.102* - file *./VOPS/files/policy/HomeComputer2.xml* is created. At the beginning this is just an empty policy, which would deny every access to this resource. User can list this policy with command

```
xlistPolicy -id HomeComputer2
```

The answer would be as shown in figure 5.

If user wants to list all VOPS' policies, he can execute next command:

```
xlistPolicies
```

and gets response as depicted in figure 6.

Response is a description of policy storage as list of all policies, which reside on that VOPS policy storage. One line contains path to policy file, policy's id and resource. As described above, policy comprises of rules, which user has to fill in. He can do that by using command

```
xaddRuleToPolicy -rid ISLsRule -pid HomeComputer2 -action
createJob -attr group -attrVal isl
```

Rule to earlier policy with id *HomeComputer2* is added as depicted in figure 7

This rule is added just before last *FinalRule* which denies every action taken. Because policy has changed, administrator has to reload VOPS policy storage with command

```
xreloadVOPS
```

It re-initializes PDP with all policy files in policy storage. After doing that, user can test, if policy permits action *createJob* with resource *192.168.2.102* to a user from group *isl*. We assume that *XOSd* and VOPS are running on a computer *192.168.2.102*. Using command

```
xverifyPolicyResNoCert -ggn isl -ugn ales -action
 submitJob
```

we get an empty list, but with command

```
xverifyPolicyResNoCert -ggn isl -ugn ales -action
 createJob
```

we obtain list *[Address = [:///192.168.2.101:60000]]*. This is resource on which user *ales* from group *isl* has a permit to execute *createJob* command.

In conclusion, next commands have been implemented:

- *listPolicies* - returns list of policy files (policy Id and resource for which this policy applies to)
- *listPolicy* - lists the whole policy file with provided policy Id
- *addRuleToPolicy* - adds rule to policy provided. Rule is described with attribute, attribute's value and action taken
- *removeRuleFromPolicy* - removes rule from distinct policy
- *verifyPolicyResNoCert* - verifies policy with no certificate present. User has to provide his name, group name and action taken. Daemon returns available (filtered) resources for that request.
- *createPolicy* - creates policy with provided parameters. File is created in policy storage. This file contains policy for resource provided.
- *removePolicy* - removes (deletes) policy from policy storage.
- *removePolicy* - removes (deletes) policy from policy storage.

API is listed below.

5.3 API of the VO Policy Service

We argued in preceding section that policies are in the folder defined in $vops.conf$ (or $./policy$ directory). This is VO Policy Service's local storage for policy files. With each VOPS command user's certificate has to be passed as an argument to verify user's credentials.

5.3.1 List policy

```
listPolicies()
```

Lists content of policy files which are stored locally by this VOPS.

5.3.2 Modify policy

- `addRuleToPolicy(String ruleId,  String policyId, String action, String attr, String groupName)`
- `removeRuleFromPolicy(String ruleId, String policyId)`

Modifies policy by adding or removing rules from or to the policy.

5.3.3 Remove policy

```
removePolicy(String policyId)
```

Removes a policy file from a local storage. Policy file is selected based on values of attributes passed to PDP. Policy with policyId will be removed from policy storage.

5.3.4 Create policy

```
createPolicy(String policyID, String description,
 String targetResource)
```

Adds policy into local storage of policies. A new policy file is created in policy storage. Policy comprises policyID, description and resource as provided with parameters. Newly created policy is shown in figure 5.

5.3.5 Request policy decision

```
verifyPolicyResNoCert(String groupGlobalName,
String userGlobalName,ArrayList<CommunicationAddress>
resources, String action)
```

Verify policies based on groupGlobalName and userGlobalName without certificate. In the future this (slightly changed function) will verify policy request based on user's certificate and resources passed to this action. It will return signed resources which will be allowed for user to use with specific command (e.g. job manager's `createJob` action).

5.3.6 Reload VOPS

```
reloadVOPS()
```

Reloads and initializes PDP module of VOPS from local policy files. Administrator should call this action after policies have changed.

6 Conclusion and Future Work

6.1 Current Status of VO Management

The current version of the prototype provides a minimal set of features to:

- *Request Credentials* - The CDA client allows an XtreemOS user to request an XOS Certificate authenticating them and containing their VO attributes from the CDA service
- *Process the VO Attributes* - A supporting Java routine allows the extraction of the user's VO attributes from an XOS Certificate
- *Present a minimal VO* - A minimal VO consisting of a small group of users, merely for testing purposes, is 'hard-wired' into the first prototype.

Whilst severely limited in scope at this time, the CDA service in this initial version of the prototype shows that many of the conceptual challenges involved in implementing the CDA have been overcome.

The XOS Certificates generated have been successfully tested for compatibility with the OpenSSL library via the use of the command 'openssl -verify'. In addition, the XOS Certificate is designed to conform to the following:

- *IETF RFC3280* - Public Key Infrastructure Certificate Profile - the fundamental format of certificates used for authentication purposes
- *IETF RFC3820 Proxy Certificate Profile* - Constraints on the above to allow short-lived certificates for delegation purposes
- *OGF Grid Certificate Profile* - Further constraints and recommendations to reflect good practice in the Grid computing field and to assist interoperability with independent implementations
- *OpenSSL Library Restrictions* - The OpenSSL library imposes some restrictions e.g. minimum and maximum length of passphrase protecting private keys, and not processing non-standard critical extensions in certificates, etc.

6.2 Current Status of VO Policy Service

Current version of VO Policy Service prototype provides these basic features:

- Access point to other services to query policies provided by *administrator* and stored locally where VOPS resides.

- Simple policy generation for testing purposes.
- Generation of requests also for testing purposes.
- Enables administrator to list, supply, delete or modify policies.
- VOPS prototype has been implemented with parallel to Application Execution Management so it's architecture enables simple integration to existing AEM prototype.

6.3 Next steps in the first prototype

Major revisions of the code will be undertaken to provide a single interface to the CDA and VO Membership functions, and to allow administration of the VO (adding and removing users and resources).

Main focus in the future for VO Policy Service implementation will be enabling dynamic generation of XACML requests and decision made by PDP module. Exact definitions for subject, resource and action segments in XACML requests has to be made.

Work on the Resource Certification Authority will be start in December. The RCA will be incorporated into the VO Management Service.

References

[1] Release of the first cda prototype. https://gforge.inria.fr/frs/download.php/3660/cda-dist.tar.gz.

[2] Release of the first vops prototype. https://gforge.inria.fr/frs/download.php/3652/aem-vops-rev1168-clean-compiled-source.tar.gz.

[3] XtreemOS Consortium. First specification of security services. http://purl.oclc.org/NET/xtreemos-d353-firstspecofsecurityservices.pdf, May 2007.

[4] XtreemOS Consortium. Second specification of security services. http://purl.oclc.org/NET/xtreemos-d354-secondspecofsecurityservices.pdf, December 2007.

[5] Simon Godik and Tim Moses. extensible access control markup language (xacml) version 1.0. http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf, February 2003.

[6] David L Groep, Michael Helm, Jens Jensen, Milan Sova, Scott Rea, Reimer Karlsen-Masur, Ursula Epting, and Mike Jones. Grid certificate profile (draft). http://purl.oclc.org/NET/OGF-CAOPS-draft-GridCertificateProfile-v25.pdf, November 2007.

[7] R. Housley, W. Polk, W. Ford, and D. Solo. Rfc 3280 - internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. http://www.ietf.org/rfc/rfc3280.txt, April 2002.

[8] S. Tuecke, W. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet x.509 public key infrastructure (pki) proxy certificate profile. http://www.ietf.org/rfc/rfc3820.txt, June 2004.

[9] Matt Welsh. Seda: An architecture for highly concurrent server applications. http://www.eecs.harvard.edu/ mdw/proj/seda/.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  PolicyId="GeneratedPolicy" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
        rule-combining-algorithm:ordered-permit-overrides">
  <Description>
    This policy applies to any accounts at users.example.com accessing
    server.example.com. The one Rule applies to the specific action of
    doing a CVS commit, but other Rules could be defined that handled
    other actions. In this case, only certain groups of people are allowed
    to commit. There is a final fall-through rule that always returns Deny.
  </Description>
  <Target>
    <Subjects> <Subject>
     <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        users.example.com </AttributeValue>
      <SubjectAttributeDesignator
        DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" />
      </SubjectMatch>
    </Subject> </Subjects>
    <Resources> <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">
            http://server.example.com/
          </AttributeValue>
          <ResourceAttributeDesignator
            DataType="http://www.w3.org/2001/XMLSchema#anyURI"
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
        </ResourceMatch>
    </Resource> </Resources>
    <Actions><AnyAction /></Actions>
  </Target>
  <Rule RuleId="CommitRule" Effect="Permit">
    <Target>
      <Subjects><AnySubject /></Subjects>
      <Resources><AnyResource /></Resources>
      <Actions> <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">commit
       </AttributeValue>
       <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
         AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id" />
      </ActionMatch>
      </Action> </Actions>
    </Target>
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <Apply FunctionId= "urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
        <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="group" Issuer="admin@users.example.com" />
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string"> developers
      </AttributeValue>
    </Condition>
  </Rule> <Rule RuleId="FinalRule" Effect="Deny" />
</Policy>
```

Figure 3: A sample policy descriptor using the XACML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="urn:oasis:names:tc:xacml:1.0:context"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <Subject>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
      DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
      <AttributeValue>seth@users.example.com</AttributeValue>
    </Attribute>
    <Attribute AttributeId="group"
      DataType="http://www.w3.org/2001/XMLSchema#string"
      Issuer="admin@users.example.com">
      <AttributeValue>developers</AttributeValue>
    </Attribute>
  </Subject>
  <Resource>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
      DataType="http://www.w3.org/2001/XMLSchema#anyURI">
      <AttributeValue>http://server.example.com/</AttributeValue>
    </Attribute>
  </Resource>
  <Action>
    <Attribute
      AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      <AttributeValue>commit</AttributeValue>
    </Attribute>
  </Action>
</Request>
```

Figure 4: A sample action request structure.

```
<Policy PolicyId="HomeComputer2" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:
        rule-combining-algorithm:ordered-permit-overrides">
  <Description>ThisIsForTestingPurposes</Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:anyURI-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#anyURI">192.
                168.2.102</AttributeValue>
          <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:
                resource:resource-id" DataType="http://www.w3.org/2001/XMLSchema#anyURI"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>
```

Figure 5: Listing of a newly created policy.

```
Policy fileName:C:\razvoj\xtreemOS\VOPS\files\policy\generated.xml; PolicyId=GeneratedPolicy;
         Resource http://server.example.com/
Policy fileName:C:\razvoj\xtreemOS\VOPS\files\policy\HomeComputer.xml; PolicyId=HomeComputer;
         Resource 192.168.2.101
Policy fileName:C:\razvoj\xtreemOS\VOPS\files\policy\HomeComputer2.xml; PolicyId=HomeComputer2;
        Resource 192.168.2.102
Policy fileName:C:\razvoj\xtreemOS\VOPS\files\policy\myPolicy1.xml; PolicyId=myPolicy1;
         Resource 192.168.0.233
Policy fileName:C:\razvoj\xtreemOS\VOPS\files\policy\newID.xml; PolicyId=newID;
         Resource localhost
Policy fileName:C:\razvoj\xtreemOS\VOPS\files\policy\test_policy1.xml; PolicyId=testPolicy1;
        Resource localhost
```

Figure 6: Response of the *xlistPolicy* command.

```
<Rule RuleId="ISLsRule" Effect="Permit">
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">createJob
              </AttributeValue>
          <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:
                      action:action-id" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
      <SubjectAttributeDesignator AttributeId="group" DataType="http://www.w3.org/2001/
              XMLSchema#string"/>
    </Apply>
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">isl</AttributeValue>
  </Condition>
</Rule>
```

Figure 7: Listing of an updated policy.