Project no. IST-033576

XtreemOS

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

Third Prototype Implementation of Security and VO Management Services
D3.5.16

Due date of deliverable: 30/03/2010
Actual submission date: 07/05/2010

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

Type: Deliverable
WP number: 3.5
Task number: T3.5.2

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 1.0 / Last edited by Ian Johnson / 07/05/2010

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	25/03/10	Alvaro Arenas	STFC	Created outline
0.1	09/04/10	Matej Artac, Aleš Černivec	XLAB	RCA and VOPS sections
0.2	16/04/10	Ian Johnson	STFC	XVOMS sections
0.2	16/04/10	Matej Artac	XLAB	Monitoring, auditing and DIXI sections
0.2	16/04/10	Zhou Zhouyi	ICT	Isolation sections
0.3	19/04/10	Ian Johnson	STFC	Introduction, general revision
0.4	23/04/10	Ian Johnson	STFC	Tackling reviewers' comments
0.4	23/04/10	Matej Artac, Aleš Černivec	XLAB	Tackling reviewers' comments
0.4	23/04/10	Zhou Zhouyi	ICT	Tackling reviewers' comments
0.5	28/04/10	Alvaro Arenas	STFC	Executive summary, conclusion, general revision
0.6	30/04/10	Yvon Jegou	INRIA	Single sign-on sections
0.6	04/05/10	Ian Johnson	STFC	General revision
0.6	05/05/10	Yan Bo	ICT	Web front-ends sections
1.0	07/05/10	Alvaro Arenas	STFC	Final revision

Reviewers:
Corina Stratan (VUA) and Christian Spann (ULM))

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.5.2	Specification of XtreemOS Security Services	STFC*, INRIA, ICT, XLAB
T3.5.3	Security Policy Management and Enforcement	XLAB*, SAP, STFC
T3.5.4	Federation and interoperability	STFC*, INRIA
T3.5.5	Trust management in Grid-based Operating Systems	INRIA*, SAP, STFC, XLAB
T3.5.7	Integration	STFC*, INRIA, ICT, ULM, XLAB
T3.5.8	VO Lifecycle Management Systems	STFC*, ICT, INRIA, XLAB
T3.5.9	Security of Grid Level Services	STFC*, SAP, ULM, XLAB
T3.5.12	Monitoring and Auditing Services	XLAB*, STFC
T3.5.13	Isolation	ICT*, INRIA

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

Executive Summary

This deliverable presents the third version of XtreemOS security and VO management services. Several new services have been developed for the third version: monitoring, auditing, single sign-on and isolation. Other services already included in the second version, such as X-VOMS, RCA, VOPS and the Web Front-ends, have been enhanced by including new functionalities.

For each service, it is included a brief description of its main functionalities, the procedure for installation and configuration, and a user guide.

The X-VOMS component provides usability improvements, allowing automation and scriptability for generating certificates. It is now possible to obtain certificates for XtreemOS core services online by using the CDA client programs. Support for revoking user certificates is also provided, allowing a Certificate Revocation List to be generated from certificate status information stored by the CDA server in the X-VOMS database.

The Resource Certification Authority (RCA) is a service that provides the means to obtain machine certificates to the resource and system nodes of the XtreemOS. We have extended the usage of RCA on multiple levels. First, the non-resource nodes can request and use machine certificates issued by the RCA. This means that services such as the RSS overlay agents can provide the VO machine certificate without being mistakenly selected in the resource selection. Further, optionally, the certificates can provide the node's IPv6 address in the CN of the subject. We have also added command-line tools that the resource administrators can use to refresh the information on their resource stored within the RCA database, and to remove the resource from the list of the registered resources (unregistration).

The Virtual Organisation Policy Service (VOPS) provides means to manage security policies in VOs as well enforce the policies at the point of resource selection process. Enhancements for the new version include refined API for users and other services, supporting filtering policies. The policy storage, eXist database, is now fully integrated with VOPS and is no longer used as separated module. Overall, the VOPS prototype is implemented more modularly.

Monitoring is a new functionality providing the mechanisms for notifying or being notified about events related to the entities in the system. With the ability to poll the state of the system or subscribe to specific events, the monitoring subsystem enables the grounds for reacting to security-critical situations.

Auditing is a new functionality for archiving of the aggregated events that can be used to trace who is responsible when security breach is attempted or when an anomaly in the system occurs. Auditing can be seeing as providing a logical extension to monitoring.

Single sign-on functionalities allow users to reuse credentials that have been

validated during previous requests without having to apply the whole validation process again. Single sign-on mainly avoids requesting users to type their password each time they use their certificate.

Isolation is a new functionality that provides the mechanism to limit job's computer resource usage according to predefined policies.

The Web front-ends are implemented to provide a one-site client service that exposes the VO and RCA related functionalities, as well as simplifies the users' requirements of accessing the core services via a web interface.

Finally, the security in DIXI, the XtreemOS communication bus, has been enhanced by including mechanisms for checking the user's credentials and blocking unauthorised use of the designated service calls.

Contents

List of Figures

Glossary

AEM	Application Execution Management
AMS	Account Mapping System
CA	Certification Authority
CDA	Certificate Distribution Authority
CN	Certificate Common Name
CRL	Certificate Revocation List
CSR	Certificate Signing Request
DIXI	Distributed XtreemOS Infrastructure
JSDL	Job Submission Description Language
NLP	Node Level Policy
PDP	Policy Decision Point
PKI	Public Key Infrastructure
RCA	Resource Certification Authority
RSS	Resource Selection Service
VOM	Virtual Organization Management
VOPS	Virtual Organization Policy Service
VOLife	Virtual Organization Lifecycle service
XtreemFS	XtreemOS File System
X-VOMS	XtreemOS Virtual Organization Management Service
XOSD	XtreemOS Daemon

1 Introduction

This deliverable presents the third version of XtreemOS security and VO management services. This introductory chapter summarises the main enhancements in relation to the second prototype, describes the standards and profiles used, and outlines the rest of the document.

1.1 Enhancements over Second Prototype

X-VOMS Enhancements The X-VOMS component provides usability improvements, allowing automation and scriptability for generating certificates. It is now possible to obtain certificates for XtreemOS core services online by using the CDA client programs ("get-service-cert"). This is the preferred method for obtaining service certificates, as it is more streamline than the previous manual, offline mechanism (which involved sending a (Certificate Signing Request) CSR by e-mail to the operator of the Root CA and receiving a certificate back by the same means. The CDA server can apply access control for obtaining service certificates at a number of levels of granularity, either allowing service certificates to be requested by all users, by specific named users, or by users who have the "GRIDADMIN" Actor attribute.

Support for revoking user certificates is now provided, allowing a Certificate Revocation List to be generated from certificate status information stored by the CDA server in the X-VOMS database. A list of CRL Distribution Points can be configured into all certificates issued.

RCA enhancements We have extended the usage of RCA on multiple levels. First, with the Third Prototype the non-resource nodes can request and use machine certificates issued by the RCA. This means that services such as the RSS overlay agents can provide the VO machine certificate without being mistakenly selected in the resource selection. Further, optionally, the certificates can provide the node's IPv6 address in the CN of the subject.

We have also added command-line tools that the resource administrators can use to refresh the information on their resource stored within the RCA database, and to remove the resource from the list of the registered resources (un-registration). Intrinsically, most of the commands are now subject to the access control checks, preventing their unauthorised use.

VOPS enhancements In the second prototype described in Security Services prototype month 36 [5] we have changed the form of the VOPS implementation in order to separate the core implementation of the service from front-end and the

database part. In order to store the policies **eXist** [10] database is used. It is used as a module and can easily be replaced with an alternative database implementation if needed. EXist provides faster and easier searching through the database. This has been tested and described in [3], where evaluation of security of VOPS and other services is provided.

Main enhancements over the second prototype of the VOPS are: better access control, refined API for users and other services to access the VOPS — support to filtering policies, eXist is now fully integrated with VOPS and is no longer used as separated module, the whole prototype is implemented more modularly.

As mentioned, access control on the server side of the VOPS is enhanced comparing to the second prototype. Roles of users from certificates are examined and only are allowed those which are defined in the VOPS configuration. Based on a role of the user access to appropriate policy storage is allowed. VO administrators must have predefined roles and these are defined in VOPS configuration file. The same goes for resource administrators and users.

VOPS API [4] is refined in a way to provide an additional parameter, namely the **complementary id**. VOPS additionally checks the ids used in commands and by that enhances security. Moreover, VOPS provides filter policies to assist with and refine the resource selection process.

Monitoring is a functionality new to the Third Prototype. It provides the mechanism for notifying or being notified about events in the caused by or related to the entities in the system. This event exchange is an important source of an up-to-date information on the behaviour of the users in the system as well as the services providing the system functionality. With the ability to poll the state of the system or subscribe to specific events, the monitoring subsystem enables the grounds for reacting to security-critical situations.

Auditing provides a logical extension to the Monitoring and is thus also new to the Third Prototype. The Auditing represents a service for archiving of the aggregated events that can be used to trace who is responsible when security breach is attempted or when an anomaly in the system occurs. Auditing, in logical sense, also provides the means to enforce VO policies.

Single sign-on is a functionality new to the Third Prototype. Single sign-on allows users to reuse credentials that have been validated during previous requests without having to apply the whole validation process again. Single sign-on mainly avoids requesting users to type their password each time they use their certificate.

Isolation is a functionality new to the Third Prototype. It provides the mechanism to limit job's computer resource usage according to predefined policies. The computer resources controlled by isolation mechanism include but not limited to: disk resource, memory resource, network bandwidth resource and cpu resource.

DIXI enhancements in terms of the security infrastructure most importantly include the mechanism for checking the user's credentials and blocking unauthorised use of the designated service calls. When the SSL is enabled for the communication, the DIXI service hosts can also use the user's certificates, which was previously not possible. This makes the boot-strapping of the SSL for the new nodes for the all-SSL communication possible. Further, the runtime of the DIXI can be pre-configured to run with a limited set of features, lowering potential security issues for the sensitive services.

1.2 Standards and Profiles Used

To lay the foundations for interoperability with other Grid middleware system, XtreemOS uses well-established Grid standards. The fundamental principles and standards for Public Key Infrastructure are described in IETF RFC3280 [9]. The XOS Certificate conforms to the Public Key Certificate profile defined in RFC3280, and also to the Proxy Certificate Profile defined in RFC3820[13]. In addition, XOS Certificate conforms to the most relevant recommendations in the OGF Grid Certificate Profile [8].

The VO Policy Service (VOPS) defines policies in the XACML 1.1 language [7]. The CDA, VOPS and RCA are first defined in [1], First Specification of Security Services, and are further refined in [4], Fourth Specification, Design and Architecture of Security and VO Management Services.

The Isolation mechanism uses Linux CGROUP framework which is contained in current Linux-2.6 kernel [11].

1.3 Deliverable Outline

The structure of the deliverable is the following. Chapter 2 gives a brief description of the XtreemOS security and VO management services. Then, chapter 3 describes the procedure for installing and configuring each of the services Finally, chapter 4 presents a user guide to the services.

2 Prototype Description

2.1 Prototype Functionality

2.2 X-VOMS

2.2.1 Overview

The XtreemOS VO Management System comprises the following components and services:

- The XtreemOS Root CDA (Root Certification Authority and Certificate Distribution Authority)
- The X-VOMS database
- The Certificate Distribution Authority.

X-VOMS provides the following functionality:

- Establishing the root of trust in an XtreemOS Grid
- Creating core service certificates
- Creating and distributing user XOS Certificates.
- Creating Certificate Revocation Lists for user XOS Certificates
- Storing user registration details and user VO membership details

New features in X-VOMS for XtreemOS release 3.0 X-VOMS now provides greater usability in the certificate signing process. The CDA Server can be configured to automatically sign authorised requests for service certificates, to allow for greater automation when setting up an XtreemOS Grid. The manual process of signing CSRs for service certificates is still available as an option. On the requester side, the CDA client can take all its parameters, including key pass-phrases, from input files, allowing for scriptability and preventing pass-phrases being exposed on the command-line.

The protocol between the CDA client and CDA server has been changed so that all parameters relating to a certificate request are included in the CSR structure itself (protecting the integrity of the parameters).

The CDA server stores information about the user certificates that it issues. This can be used to generate a Certificate Revocation List in case a user's private key is lost or compromised, or when a user is removed from a VO. In addition, user certificates can be configured to contain a list of CRL distribution points from where CRLs can be downloaded.

2.2.2 The XtreemOS Root CDA

The XtreemOS Root CDA comprises two functions, the Root CA and the CDA server. The Root CA holds the root of trust in the system, comprising the Root CA private key and public key certificate. The CDA server is used to sign certificates for users and for core services.

The Root CA is used during the creation of an XtreemOS Grid to sign the service certificate for the CDA server; by this means, the Root CA delegates trust to the online CDA server. The Root CA function is inactive except during the initialisation of the Grid, and when the CDA service certificate needs to be re-generated. The Root CA private key can, and should be, held offline after the Root CA has signed the CDA service certificate. Ideally, the Root CA would be operated offline (on a machine not connected to the network), with the Root CA public key certificate transferred to the Root CDA by a manual means (such as a removable memory device). A less secure option, which might be appropriate for some environments, is to keep the Root CA on a machine which is heavily firewalled, not accepting connection requests.

The CDA server component of the Root CDA handles two types of certificate: user XOS-Certificate and core service certificates. In both cases, users need to authenticate to the CDA server with their username and password. For issuing service certificates, the CDA server can be configured to perform authorisation checks on the requestor of the service certificate (either allow everyone, allow named users, or allow users who have the 'GRIDADMIN' Actor status in the Grid).

Users can alternativley use the offline, manual method of submitting requests for core service certificates manually to a Grid administrator who has access to the CDA server. Such a Grid administrator can inspect the request and run a manual script to create a service certificate signed by the CDA private key. This mode of working, however, is deprecated in favour of using the "get-service-cert" command.

The CDA service certificate is sent to the CDA client during the SSL hand-shake; this allows a client node connecting to such a service to authenticate the service whilst establishing an SSL connection. The client node has access to a local copy of the XtreemOS Root CA certificate, which contains the public key matching the Root CA private key. This allows the CDA client to verify that the CDA service certificate was issued by the Root CA and, hence, can be trusted.

2.2.3 The X-VOMS Database

The X-VOMS database holds registration information about users (such as their username, real names, expiry date, host organisation etc), and information on VOs

(such as VO names and identifiers, roles and groups).

Database entities The X-VOMS database uses the Hibernate Object-Relational Mapping library to support a flexible VO model. Entities in a VO are identified by Universally Unique Identifiers [1] (UUIDs), and also short and long descriptions for convenience in displaying information to the user.

Types of database entity The entities can be divided into two categories; user registration details and VO details.

2.2.4 The Certificate Distribution Authority

The Certificate Distribution Authority provides a means for users to obtain an XOS-Certificate which contains their VO identity, and their VO attributes defining which VOs and groups they belong to, and which roles they hold. The CDA is implemented by a server which accesses the X-VOMS database.

CDA Protocol The CDA server processes requests from the CDA client programs, either "get-xos-cert" or "get-service-cert". The communication uses a secure channel and takes the form of a protocol with two commands. The user's username and password are sent with the `AUTHENTICATE user,password` command. The response to this is `AUTHOK` if the details are verified, or `FAILED reason` if the user cannot be authenticated. The request for a certificate is sent with the command: `CERTREQUEST <CSR>`. The CSR structure contains the user's public key and an optional field requesting a non-default certificate lifetime. Additional request fields in the CSR vary depending on whether the request is for an XOS-Certificate or a Service Certificate.

These variant extensions are as follows:

Certificate Type	Extensions in CSR
XOS-Certificate	voName, groupName
Service Certificate	hostname, service type, description of service

2.2.5 Operation of the CDA Client

The CDA client program, `"get-xos-cert"`, is used whenever the XtreemOS user needs to obtain an XOS-Certificate. This contains their VO attributes, and normally lasts for 30 days. The other form of the CDA client program,

[1] en.wikipedia.org/wiki/Universally_Unique_Identifier

"`get-service-cert`" is used to obtain service certificates, and operates in a similar manner.

Establishing an authenticated connection between the CDA client and CDA server The CDA client establishes a secure connection to the CDA server using SSL. As part of the SSL handshake, the CDA server sends the client a certificate chain consisting of the CDA's service certicate and the XtreemOS Root certificate. After the SSL handshake has completed, a "post-connection verification" step takes place; thiscomprises a set of checks which augment the very minimal checks performed automatically during the SSL handshake.

The client verifies the connection by checking that the hostname encoded in the CDA's service certificate is the same as the hostname it is connecting to, and that the CDA's service certificate has been signed by the private key corresponding to the public key contained in the CDA service certificate. Finally, the CDA client checks that the Authority Key Identifier field in the CDA service certificate corresponds to the client node's local copy of the XtreemOS Root CA certificate. If all of these tests succeed, the CDA client can be confident that it is connecting to the authentic CDA server for this XtreemOS Grid. This is important, as the CDA client needs to protect the user's username and password, which are sent in the next step.

Authenticating the user to the CDA server The CDA client prompts the user for their username and password, and sends the `AUTHENTICATE` command to the CDA server. If the user is authenticated successfully, the next step is executed, otherwise the CDA client prints an error message to the user and exits.

Creating a private key and Certificate Signing Request (CSR) in the CDA Client The CDA client can create a new public/private keypair for the user, or use an existing keypair. The latter option is more efficient for resource-limited nodes, and is also useful where an existing keypair resides in a JKS (Java KeyStore) file. If a new keypair is created, the user is prompted for a secret pass-phrase to protect the keypair, and the keypair is stored in a file readable only by the user. If an exisiting keypair is specified, the user is prompted for the passphrase protecting the keypair. Optionally, the CDA client can read an existing Certificate Signing Request, which may be generated by a different program (such as Java "keytool" or OpenSSL "openssl req").

Sending the CSR to the CDA server If the CDA client creates a new Certificate Signing Request, this contains the public part of the user's key, the voName and

groupName. If the user specifies a desired duration for the XOS-Certificate, this is encoded as an extension field in the CSR. The CSR is then signed by the user's private key, which allows the CDA server to establish the authenticity of the public key that is contains. The CSR is then sent to the CDA in the `CERTREQUEST` command.

Receiving the XOS-Certificate or Service Certificate When the CDA client receives the XOS-Certificate/Service Certificate in the response to the `CERTREQUEST` command, it verifies that the certificate has been issued by the CDA by checking that the signature on the XOS-Certificate is valid. If this succeeds, the CDA client stores the XOS-Certificate in the user's filestore.

2.2.6 The CDA Server

Processing a connection request When the CDA client opnes a connection to the CDA server, the CDA server sends the CDA service certificate to the client.

Authentication The CDA server authenticates a user `AUTHENTICATE` request by checking that the username and password supplied match those stored in the X-VOMS database, and that the user's registration has been approved and has not expired. If any of these details do not match, the user request is rejected.

Handling certificate requests The CDA server processes a `CERTREQUEST` by firstly checking that the user is a member of the VO specified as the primary VO. If this is the case, the user's public key is extracted from the CSR structure and used to initialise a new XOS-Certificate for the user. If the user has requested a specific validity period for the certificate (lower than the default validity period), the CDA server sets the certificate `notAfter` date appropriately.

Encoding VO Attributes in the XOS-Certificate Every user in the system has a GUID for their registration ID. This used as the value of the certificate Subject:CN field and for the VO attribute GlobalUserID. The VO ID relating to the primary VO name is used for the attribute GlobalPrimaryVOName. The CDA server nexts retrieves the list of VOs that the user belongs to from the X-VOMS database. Any VO IDs apart from the primary VO are put in a list of secondary VOs. Similarly, all groups that the user belongs to apart from the primary group are put into a list of secondary groups. The values of the GlobalPrimaryName, GlobalVOName, GlobalPrimaryGroup and lists of secondary VOs and groups are then used as the values for certificate extension fields.

Recording issued certificates The CDA server writes a record of each XOS-Certificate it issues, consisting of the certificate's serial number, the Subject:CN field, the starting and end dates for the certificate, and the VO Identifier. If a certificate is revoked, the revocation date is stored in this record and the certificate status set to "revoked".

2.3 Description of RCA Functionalities

The **Resource Certification Authority (RCA)** is a service that provides the means to obtain machine certificates to the resource and system nodes of the XtreemOS. The distribution of the trusted certificates enables the XtreemOS nodes the proof of their trustworthiness.

The primary purpose of the RCA is to be the contact point between higher level administration in the XtreemOS and the resource administrators (and the resources these administrators are in charge of). We have described the detailed use-cases of the RCA in the previous deliverable [4], thus we will only provide a summary of the functionality here.

The RCA contains a database of the resources and their end-points (addresses) that can be can be considered trustworthy. This is achieved through a process which involves the following steps:

- The resource administrator who is adding a new resource to be used in the XtreemOS, issues a command-line tool which publishes the resource's details to the RCA. These details are only visible to the site administrator.
- The site administrator can decide which candidate resources are trustworthy, and can grant or deny a resource to be addded to the trustworthy resource list. This effectively registers the resource.

Any resource that is registered with the RCA can then obtain a machine certificate signed by a trusted certificate authority. This authority is provided by the RCA which, in turn, proves its trustworthiness with its service certificate signed either by the CDA certification authority or directly by the root authority. Further, the resources or services in the XtreemOS can also use RCA for obtaining the public certificates of other resources.

The RCA can issue several types of certificates to the machines and can therefore provide a basis for many types of usages, be it built into the XtreemOS or custom.

- The **machine identity certificate** is the main certificate issued by the RCA. This certificate proves to the peers that (by being able to sign the communication using the private key counterpart) the resource belongs to the trusted domain.

- The **machine attribute certificate** extends the usage to form a credential, proving that the resource supports capabilities listed in the certificate extensions.

- The **machine VO certificate** proves that the resource or the services using the certificate act within the given VO.

- The **service certificates** are a special case of machine attribute certificates, proving capability to host a service defined in the certificate. Each certificate contains the credential of only one service.

Until the 3rd release of the XtreemOS, the usage of the machine attribute certificates and the machine VO attributes were mainly within the AEM, supporting the resource discovery, and in the interaction between AEM and VOPS, also provided the policy information point the means to obtain the relevant information on the resource. In the 3rd release, the usage was extended to support core nodes and their services by giving the choice to omit the attributes related to job execution. In this way, the services and nodes could prove that, e.g., they belong to a certain VO overlay, but at the same time express that they do not expect to take part in the resource discovery.

The RCA is essentially a core service deployed centrally within an organisation. On top of the registered resources it also maintains the list of VOs the RCA is contributing to, and also the subset of the registered resources members of the VO. The service exposes the API towards clients such as the VOLife web application. Logically, however, RCA includes a service deployed on each node. This client-side service connects to the core service's API and performes node-specific tasks such as obtaining the information on the node, generating private keys and saving certificates in a safe location. Installation of the RCA client is complete with the command-line utilities that the resource administrator can use for the interaction with the RCA.

2.4 Description of VOPS Functionalities

The **Virtual Organisation Policy Service** (VOPS) is a one per VO core service serving requests to other VO services and VO users. It provides means to manage security policies in VOs as well enforce the policies at the point of resource selection process. VO services interacting with VOPS are manly those which take part in resource selection process. Detailed use-cases, VOPS capabilities and VOPS design is available in the deliverable [4].

It provides policy administration point for VO administrators, resource administrators and VO users. As administration point we refer to ability to administer

policies for which the administrator or user is responsible for. VO administrators is able to administer (list, add, remove) policies relevant to the VO. Resource administrators can administer policies applying to the resource which the administrator is responsible for. Users administer their own policies. Owner of the policies is always represented by the user identity certificate. However, **machine identity certificate** (see section 2.3) is required with administration of the resources to identify the resource in question.

For such a service it is essential to provide proper and effective policy administration, effective operations in policy filtering, be scalable and able to serve multiple request effectively. For this purpose, the DIXI 2.10 framework already used in several XtreemOS services has been used for the implementation of the communication part. The core of the VOPS service is separated from front-end running inside DIXI.

VOPS database consists of three separated policy storages: VO, resource and user storage. User id accessing VOPS API is checked inside the VOPS front-end by inspecting the user certificate. It checks whether the role inside certificate conforms to those allowed from the VOPS configuration (see section 3.3). After initial set-up, each storage contains **default** policy enabling any action on any resource to any subject. User interacting with VOPS can list, add or remove the policies belonging to the user or her domain inside the storage which depends on the role specified inside user certificate.

Most noticeable change in the third version of VOPS is the usage of the XML database eXist [10]. This change has been made in the back-end of the service. It increases simplicity with handling XML policies, namely searching through policies for distinct value of attributes and easier construction of filter policies. It also provides efficient way to store XACML [12] policies and provides XQuery processing for easier information extraction from policies stored in the database.

2.5 Description of Front-Ends Functionalities

In XtreemOS, security services X-VOMS, RCA and VOPS are interacted using small programs or client services(e.g. RCA client), with which aside from knowledges of the services themselves a user needs explicitly be engaged in tedious tasks such as manually acquiring the certificates, and setting up service bootstrap configurations. In these cases, the high knowledge requirement and learning curve impede the usability of core services and thereby limit their widespread acceptance. Besides the client programs of the services, web front-ends are implemented to provide a one-site client service that exposes the VO and RCA related functionalities, as well as simplifies the users' requirements of accessing the core services via a web interface.

In the third release of XtreemOS, web front-ends are splited into two parts: VO

web front-end for VO management, RCA web front-end for RCA and resource management.

2.5.1 Functionalities provided by VO web front-end:

Identity management This is the interface to identity-related services which provides functionalities including grid user sign up, PKI key pair generation and download, and XOS certificate signing and maintenance.

VO management This is a essential part that enables a grid user to create, control its own VOs, and request to join/leave other VOs without ownership in order to provide or share resources.

2.5.2 Functionalities provided by RCA web front-end:

RCA and resource management This provides capabilities of RCA management, including registering RCA, adding/deleting resource to RCA and VO, approving resource adding request.

2.6 Description of Monitoring Functionalities

Monitoring is a service which provides the means for gathering monitoring data about XtreemOS. The data is collected when various services send events and metrics to the Monitoring Service. Services can create monitoring rules which describe when to trigger notifications e.g. when a user tried to use an expired certificate. Interested services can subscribe to monitoring rules and get notified. Each monitoring rule is identified by a name that is assigned when creating a rule.

Monitoring can be used to track security related type of information about XtreemOS such as user certificate expiration, tracking of certain executed commands, etc. It uses an internal in-memory database to temporarily store received metrics and events. When Monitoring Service receives an event or a metric, it checks if any monitoring rule's conditions are satisfied and if they are, then a callback is triggered and subscribes get notified.

The data is stored for a brief period of time and gets garbage collected after a while depending on the configuration of the Monitoring Service. To permanently store the data then Auditing Service must be used 2.7.

To make monitoring distributed, the Monitoring Directory Service must be used. The Purpose of the Monitoring Directory Service is to keep track of each Monitoring Service instance and to update all instances when new monitoring rules or subscribers are added or removed. There can be many Monitoring Directories instances running at the same time, which usually is one on every core-level

node. The Monitoring Service communicates with it's nearest Monitoring Directory Service (and vice versa) which is provided by DIXI. Data replication is not (yet) supported.

VOPS and Monitoring Service provide a way to monitor VOPS policies that are set by different administrators. When VOPS receives a request for new policy, it checks if there are any monitoring rules that can be extracted and every new monitoring rule is then added to Monitoring Service. A list of policy monitoring rules can be accessed by any service that is interested in such rules. For example, AEM might be interested in a rule that is triggered when a number of running jobs exceeds the number of allowed running jobs. In this case the AEM might want to cancel the execution of the extra running jobs.

2.7 Description of Auditing Functionalities

While Monitoring Service is used for gathering monitoring data, Auditing Service is used to permanently store the data which can be queried later. The data is stored in a relational database with Hibernate on top to provide Object-Relational Mapping (ORM). It's a centralized service meaning there should be only one instance of Auditing Service running. In order for Auditing Service to work, at least one instance of Monitoring Service must be running.

Auditing Service sets archiving rules at startup to start receiving various security related notifications e.g. usage of expired certificates, certain security related commands, etc. Additional rules can be added later on. Auditing rules is just another name for monitoring rules that are referring to Auditing Service.

Based on archived data, analyzes, statistics and reports can be generated. Querying of the Auditing database is done using Hibernate Query Language (HQL) and because we have direct access to the database, there is a lot of flexibility in the type of data we want to retrieve, though we must know the structure of the database. Hibernate also supports aggregation operations such as average, maximum and count which makes statistical reports generation even easier.

At the moment Auditing Service is not distributed and therefore only one instance of the Service must be running.

2.8 Description of Single Sign-On Functionalities

User credentials in XtreemOS are stored in a document (file), the user certificate. Each time this document is transmitted from user space to some XtreemOS service, it must be validated. One of the validation steps consists in checking that this document is really owned by the user and has not been stolen. Using X.509 certificates, the user must participate to a key challenge using his private key. This generally involves typing some password. Using single sign-on functionality, a

user needs to type his password and to participate to the key challenge only once during a session.

The XOS-SSO service developed for XtreemOS handles all interactions between user space and the grid service space of XtreemOS. This service runs on some XtreemOS core node and is trusted by XtreemOS grid services. It is in charge of validating all user credentials transmitted inside grid requests from the user space. In order to provide single sign-on, this service can validate and store the credentials when they are provided by the users and piggy-back these credentials inside later grid requests. The XOS-SSO service can be replicated on the resource nodes allocated for a job and so allow the job process to make grid requests on behalf of the user (delegation).

The XOS-SSO service authenticates requests generated by local processes (processes running on the same node) using the local UNIX credentials. The service also provides means to untrusted nodes to access a grid system through a specific SSH-XOS subsystem.

2.9 Description of Isolation Functionalities

Due to the limited availability of physical or logical components within a resource node of XtreemOS versus possible infinite resource requirement of job execution, resource control and management are inevitable. We introduce here resource isolation functionalities into security and VO services. Type of resources in local resource node includes and is not limited to: CPU resource, memory resource, disk space and network bandwidth.

Resource isolation functions are implemented by means of Linux Control Group (cgroup). Linux Control Group is a framework that divide system processes into resource control groups. The processes belong to the same cgroup share the same set of limited computer resources.

The isolation functions are performed according to following procedure: AEM (Application Execution Management) will get a submitted job done by invoking local resource node's AMS (Account Mapping System) subsystem. AMS subsystem receive the low level job description from AEM including the executable file name (/bin/ls for example), executing parameters (-l for example) and resource quotation information. AMS will create a process to perform the task as requested by AEM. But before performing any concrete job, AMS should create a resource control group, assign above process to that group and set up the group's quotation according to the resource quotation information provided by AEM.

The isolation is a built-in functionality that is not directly available to the end user. The actual usage of the isolation thus depends on the factors such as the resource scheduler strategies and reservation time-tables. The user mostly effects

the isolation with the resource requirements as expressed using the standard Job Submission Description Language (JSDL).

For demonstration purposes, however, we extended the JSDL schema to let the user select custom values for the isolation. The following extract from the JSDL is an example showing such a JSDL. Please note that the system will override the values set by the user according to the JSDL values, the end resource capabilities, and any policies.

```
<?xml version="1.0" encoding="UTF-8"?>
 <jsdl:JobDefinition
        xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
        xmlns:jsdl-xos="http://schemas.ggf.org/jsdl/2005/11/jsdl-xos"
        xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix">
    <jsdl:JobDescription>
        <jsdl:JobIdentification>
            <!-- your job's identification here -->
        </jsdl:JobIdentification>
        <jsdl:Application>
            <!-- the application, binary, parameters -->
        </jsdl:Application>
        <jsdl:Resources>
            <!-- resource requirements -->

            <jsdl-xos:Isolation>
                <jsdl-xos:MemorySize>1073741824</jsdl-xos:MemorySize>
                <jsdl-xos:DiskSpace>20971520</jsdl-xos:DiskSpace>
            </jsdl-xos:Isolation>
        </jsdl:Resources>
    </jsdl:JobDescription>
 </jsdl:JobDefinition>
```

2.10 Description of DIXI

The Distributed XtreemOS Infrastructure [6] is a general purpose middleware and a messaging bus developed in XtreemOS. It features the ability to stage the services and host them in a distributed manner. The DIXI system provides layers that automatically translate the Java service method invocations into service messages and sends or receives them using TCP. It provides support for the SSL, and the currently built-in message passing protocol used on top can be extended or supplemented with any other networking protocol. This abstraction of the service calls from the actual messaging transport provides portability of the services and, through the in-built service discovery facility, a truly dynamic and distributed system of services.

Previously [2] we have provided the security evaluation, which we based on the architectural properties of the DIXI. Considering it is a new infrastructure, the off-the-shelf probing tools for identifying security weaknesses found no issues with the system running a DIXI. However, DIXI uses a service message passing

protocol with the messages represented in Java classes or using an XML translation. The implementation is open, making it simple for any third-party software to successfully invoke the service calls. Hence, we built into the final prototype certain enforcements that would prevent any unauthorised users to invoke sensitive actions.

The DIXI supports two types of networking channels: plain connection using TCP/IP, and the encrypted (SSL) communication. There is currently a technical restriction that all DIXI hosts should use the same type of communication, though the mixing could be supported. The plain sockets are provided for the simplicity and the testing purposes, however on the production nodes it should never be used unless the grid is guarded from external access and untrusted personnel by some other means (such as firewalls). It also leaves the services to implement their own encryption and security techniques, which would remove the abstraction between the service implementation and the interchangeable communication layer.

When the DIXI nodes engage in the SSL communication, a considerable level of security is applied instantly. Any standard certificates can be used in the SSL handshake, as long as the signer of the certificate is trusted by the communication peer. By default this includes any certificate signed by a trusted RCA as well as the user certificates signed by a trusted CDA. In practice it has turned out that it is the most simple to use the machine certificate and its private key, because RCA is trusted by all parties. The problem arises when a new node needs to be added into the network, because in order to obtain the certificate from the RCA, the client already needs to use a trusted certificate. We solved the issue by enabling the user's certificates to be also valid in the SSL handshake, which is not usually possible because of their in the clients only. Since this certificate needs to be obtained through an SSL communication in the first place, the administrator can temporarily configure the DIXI on the node to use the administrator's user certificate for the handshakes.

The services staged within a DIXI instance share the memory space with other co-located services. This also means that the DIXI instance will communicate with other instances using the global communication settings and the SSL certificate. Often the services need a finer grain of authentication. In such a case the services in their calls accommodate for a credential in a formal parameter. The service can then implement the logic for checking the SSL peer certificate, provided by the DIXI API, and the certificate passed directly in the call. To avoid duplication of work, however, DIXI provides a generic mechanism for performing access control checks for the service calls. The credentials take form of the XtreemOS's standard user certificates, using the extensions denoting the user's group and role. Engaging the access control in DIXI thus becomes the matter of adding a decoration to the service call implementation in the design time. The developer assigns each restricted service call a category, which will be associated

with the roles and groups of users permitted to invoke the service call. As a result, this scheme prevents any unauthorised users from invoking any part of the user code within the service call.

The DIXI contains many features active in the runtime that are designed to ease the development of the services. This includes service message redirection when the receiving host is unable to serve the request and automatic service look-up using the peer-to-peer overlay. When we need to host sensitive services, however, such features may be undesirable because they could unintentionally provide ways to abuse them in order to gain unauthorised access to restricted features. To provide support for such services, the DIXI can be configured to disable service look-up and message call redirection. Instead, it obtains the addresses of the needed services from a static configuration file.

The client programs normally take advantage of the XATI library, which contains a small subset of the DIXI service hosting program. This means that it contains a small message bus and a set of pre-generated modules that, all together, take care of the communication between the user's program and the service hosts. To be able to use the DIXI service API, however, the client does not have to use the XATI library. In case of strict and tight security it is therefore possible to use entirely custom routines for communication. The downside of this choice, however, is the extra work incurred by implementing the communication routines and the need to follow the changes in the service's API.

2.11 Prototype Architecture

The VO model in XtreemOS supports multiple VOs co-existing in a single XtreemOS Grid. The third prototype of the XtreemOS Security Services places the following constraints on the security and VO management services in terms of number of instances and location of components:

Service	Instance count	Placement
X-VOMS database	single	core node A
CDA Server	single	core node A
VOWeb	single	core node A
RCA Server	1 instance per administrative domain	core node
RCAWeb	1 instance per administrative domain	code node
VOPS	single	core node

From this, we can see that only single instances of the X-VOMS database, CDA server, VOWeb web application and VOPS server are allowed in any XtreemOS Grid.

Placement of components The CDA server and VOWeb components access the same database, and are only supported running on the same host as the X-VOMS database server (there are no database connections allowed to the database server from other nodes).

2.12 Prototype Deployment

Placement of components The X-VOMS and VOWeb components access the same database, and must be co-located on the same host as the X-VOMS database.

Minimal Configuration For a small-scale deployment, it is possible to run all the security components on the same core node. This does, however, limit the resources available in this Grid to those residing in the same organisational domain (this is because the RCA server only trusts requests originating from its own domain).

Common Configuration A more common way of deploying the security components is to co-locate the X-XVOMS and VOWeb components on the same host that is running the X-VOMS database, and choose another core node to host the VOPS service. Each organisational domain providing resources to the Grid needs to run its own instance of the RCA server.

3 Installation and Configuration of Security Services

This chapter describes how to install the constituent security services and how to configure them.

3.1 Installation and Configuration of X-VOMS

3.1.1 X-VOMS Root Certification Authority and Certificate Distribution Authority

The Root CA is the top level of the trust mechanism in XtreemOS. It is a critical part in the XtreemOS Public Key Infrastructure (PKI). To achieve and maintain the level of trust required by users of an XtreemOS Grid, the Root CA must be operated only on one machine. This host must be a physically-secure core node to avoid compromise of the Root CA private key, which would destroy any trust placed on the Root CA. Some organisations may choose to run the Root CA on a machine which isn't connected to a network, to eliminate any risk of intrusion.

The Root CA comprises root entity credentials which are trusted by all participants in an XtreemOS Grid, and an optional mechanism to manually create certificates for XtreemOS core services, if this task isn't being performed automatically by the Certificate Distribution Authority.

The package `certificate-utils` contains the configuration files for creating a Root CA, and for optionally creating service certificates from Certificate Signing Requests.

The first step is to install the `certificate-utils` package. This places configuration files in `/etc/xos/config/openssl`. Decide on a directory to hold the files related to the Root CA, for example, `/opt/xtreemosca`.

The Root CA certificate is configured by settings in the file `/etc/xos/config/openssl/create-rootca-creds.conf`. The section `[ root_ca_distinguished_name ]` can be modified to change the certificate fields `commonName`, `organizationName` and `organizationalUnitName` as required. The `[ req ] section` contains the property `default_days` to set the duration of the certificate's validity, and `default_bits` to set the size of the Root CA private key.

The next step will create the Root CA directory, and public and private key for both the Root CA and CDA server. You need to pass on the command line the directory you have chosen for the Root CA directory, the hostname of the server that will run the CDA service, and a brief description for the CDA service.

To create these credentials, run the command shown in Figure 1.

```
# create-rootca-cda /opt/xtreemosca cdahostname 'description'
```

Figure 1: Creating the XtreemOS Root CA and CDA credentials.

You will be prompted first for a passphrase to protect the Root CA private key. This passphrase must be kept secret to prevent use of the private key by anyone other than the operator of the Root CA.

The private key is created in the sub-directory `private` under the Root CA location (in this case, `/opt/xtreemosca/private/xtreemos.key`). The public key certificate of the Root CA is the XtreemOS 'root certificate'. It is created in the sub-directory `public` of the Root CA directory (in this example, `/opt/xtreemosca/public/xtreemos.crt`). The XtreemOS root certificate needs to be installed on all machines in this XtreemOS Grid. The certificate can be placed in
`/etc/xos/truststore/certs/xtreemos.crt` on these machines.

You will next be prompted for the passphrase for the CDA private key. and then prompted for the passphrase for the Root CA private key which signs the CDA public key certificate. The script will create the CDA private key in the file `/opt/xtreemosca/private/cda.key` and the CDA public key certificate in `/opt/xtreemosca/public/cda.crt`. The CDA public key certificate can be distributed to all machines in the Grid, along with the Root CA certificate, to allow verification of the full certificate chain for certificates issued by the CDA. However, the CDA certificate is also stored by the CDA client when it requests a certificate.

Once the XtreemOS root certificate has been distributed, the Root CA can be moved offline or the machine holding it can be turned off.

Creating service certificates Certificates for the core services can be obtained from the CDA server with the "get-service-cert" command as described later.

3.1.2 Configuring the X-VOMS database

X-VOMS (XtreemOS Virtual Organization Management Service) is an advanced Virtual Organisation (VO) management service for supporting secure and flexible collaborations and resource sharing among people, projects and organisations. It is written in Java and backed by a (Hibernate-based) X-VOMS database schema. Like other VO management software packages, X-VOMS provides a set of APIs for managing identity, attributes, and VO membership of users and resources.

X-VOMS manages a user's VO attributes. It is used in conjunction with the Credential Distribution Authority (CDA) to disseminate credentials.

Software prerequisites The current X-VOMS implementation relies on the following software:

- Hibernate 3.0 [2] (or newer)
- MySQL 5.1.6[3] (or newer)

Major files and their location The steps needed to create the X-VOMS database and load it with data are encapsulated in the script xvoms_init.sh:

Configure the X-VOMS database, as illustrated in Figure 2.

```
# /usr/share/xvoms/bin/xvoms_init.sh
```

Figure 2: Configuring the X-VOMS database.

This command prompts the user to set a password for the X-VOMS database 'root' user when run for the first time. If the command is re-run, the command will prompt for the database root password.

The command then enables the MySQL database to accept connections over TCP/IP by editing the file `/etc/my.cnf` file, and loads the X-VOMS database schema and some example data.

Running the `xvoms_init.sh` script is sufficient to allow the following steps 'Installing the CDA' (3.1.3), 'Installing VOLife' (**??**) etc to be performed.

[2] http://www.hibernate.org/

[3] http://www.mysql.com

The following files described below are merely described for reference purposes as their functionality is incorporated in the xvoms_init.sh script.

The configuration files are located at: `/usr/share/xvoms/`. The X-VOMS library (xvoms-version.jar) is located at: `/usr/share/java/`.

- /usr/share/xvoms/scripts/setup.sql

  This script sets up the basic X-VOMS table schema. This is the first step performed by the xvoms_init.sh script. Without setting up the tables, some security features (such as X-VOMS access control and authentication) cannot be demonstrated or your requests will be automatically denied (Figure 3).

```
# mysql -u root --password=root_pass < setup.sql
```

Figure 3: Initialising the X-VOMS database.

  This command populates three tables: rules, actors, and actions, which are essential for X-VOMS.

- `/usr/share/xvoms/data/xvoms.txt`

  a sample xvoms database file, including both schema and some sample data (users, vos, vo attributes). To use this file to setup a sample xvoms database, perform the steps presented in Figure 4.

```
Root # mysql -u root --password=root_pass
mysql> create database xvoms;
mysql> quit;
mysql -u root --password=xxxxx xvoms < \
/usr/share/xvoms/data/xvoms.txt
```

Figure 4: Populating the X-VOMS database.

  An example account `xtreemos-vouser`, is included in the file `xvoms.txt` to allow testing X-VOMS. The password for this account is 'xtreemos'.

  To refresh the sample file, you can do as shown in Figure 5.

  This command will prompt for the root password of the X-VOMS database, or the root password can be passed as the second argument to the command.

```
# /usr/share/xvoms/bin/xvoms_load.sh /usr/share/xvoms/data/xvoms.txt
```

Figure 5: Refreshing the X-VOMS database.

- `/etc/xos/config/xvoms/hibernate.cfg.xml` a hibernate configuration file for setting hibernate connection properties. The most notable settings are:

```
<property name="connection.url">
jdbc:mysql://localhost/xvoms
</property>
<property name="connection.username">volifecycle</property>
<property name="connection.password">xosvo</property>
```

XtreemOS only supports running the X-VOMS database on the same host as the services that access it, hence `localhost` is specified in the connection URL.

3.1.3 Configuring and Running a Certificate Distribution Authority (CDA) Server

This sub-section is for a Grid administrator running a CDA server. This section should be performed after the sections 3.1.1 and 2 have been carried out.

The Certificate Distribution Authority is implemented in the **cdaserver** package. The CDA server runs on a single core node in an XtreemOS Grid.

The standalone CDA client program communicates with the CDA server, and is provided by the **cdaclient** package.

The CDA server issues user XOS-Certificates and, optionally, service certificates. The server needs an service certificate issued by the Root CA to authenticate itself to the corresponding CDA client. This CDA service certificate is generated when installing the Root CA, as described in 3.1.1. This procedure also produces a private key, which should be placed into `/etc/xos/truststore/private/cda.key`. The service certificate contains the service's public key, and should be placed in `/etc/xos/truststore/certs/cda.crt`.

As root, install the CDA server as shown in Figure 6.

```
# urpmi cdaserver
```

Figure 6: Installing the CDA server.

The following aspects of the CDA server are configurable by setting values in the file `/etc/xos/config/xvoms/xvoms.properties`:

- **cdaserver.keyFilename** — private key of CDA server - must be kept secure, readable only by owner.
- **cdaserver.keyPassphrase** — the private key is secured by a passphrase, the longer the better.
- **cdaserver.certFilename** — public key certificate of CDA server.
- **xtreemos.rootCertificate** — public key certificate of root CA.
- **cdaserver.sslAlgorithm** — cipher used by SSL.
- **cdaserver.sslHandshakeCipher** — the cipher used in initial SSL key exchange.
- **cdaserver.signatureAlgorithm** — algorithm used to sign the XOS-certificate returned to user.
- **cdaserver.validityDays** — number of days that certificate is valid for
- **cdaserver.validityHours** — number of hours that certificate is valid for
- **cdaserver.rejectUserCSRs** — boolean specifying whether requests for user XOS-Certificates wil *not* be signed (default is to sign requests for XOS-Certs)
- **cdaserver.acceptServicerCSRs** — boolean specifying whether requests for service certificates wil be signed (default it *not* to sign such requests)
- **cdaserver.allowAnyoneServiceCSRs** — if acceptServiceCSRs is true, specify whether any user can request a service certificate
- **cdaserver.usernamesForServiceCSRs** — if acceptServiceCSRs is true and allowAnyoneServiceCSRs is false, specify a comma-separated list of usernames that are authorised to request a service certificate
- **cdaserver.allowActorGridAdminForServiceCSRs** — allow users who have the "GRIDADMIN" Actor status to request service certificates
- **cdaserver.CRLDistPoints** — a comma-separated list of URIs from where Certificate Revocation Lists can be obtained to check the revocation status of XOS-Certificates. This can include a mixture of file: and http: URIs.

- **cdaserver.validityDays** — default number of days that certificate is valid for
- **cdaserver.validityHours** — number of hours that certificate is valid for
- **cdaserver.validityMinutes** — number of minutes that certificate is valid for

The default validity of a certificate is calculated as (cdaserver.validityDays) days + (cdaserver.validityHours) hours + (cdaserver.validityMinutes) minutes. Any two of these values can be zero. Hence, the lifetime of certificates issued by the CDA server can be set on a fine basis, if required. The CDA server will create a certificate with a shorter lifetime than the default if the CDA client program `get-xos-cert` or `get-service-cert` is invoked with the '-D days' argument.

Other aspects of the CDA server operation are:

Connection to X-VOMS database - this is set in `/etc/xos/config/xvoms/hibernate.cfg.xml`.

The level of logging, log file location, etc, are defined in `/etc/xos/config/cdaserver/log4j.properties`.

Once configured, the server is started by issuing the command shown in Figure 7.

```
# /sbin/service cdaserver start
```

Figure 7: Installing the CDA server.

The server writes its log files in `/var/log/cdaserver/cdaserver.log` by default.

Most service certificates are used to authenticate core services to client programs. For mounting XtreemFS filesystems, one mode of use is to use the service certificate for the `xtfs_mount` application to authenticate the client host to the XtreemFS server. Alternatively, the XtreemFS mount client can also use an XOS-certificate if the client is being run on behalf of a single user.

Prerequisite for installing any core service application. The following conditions apply:
Before installing any server, the XtreemOS Certificate Distribution Authority must be active in your XtreemOS Grid. See Section 3.1.3.
The "get-service-cert" command should then be used to obtain a service certificate from the CDA server.

Optional: Deprecated manual method of creating service certificates The following method is no longer recommended, as the CDA server should be used to create service certificates automatically.

The `create-csr` package must be installed; it contains the **create-csr** command and an OpenSSL configuration file to create a certificate signing request (CSR) file for an application. The requestor must then send the CSR to the operator of the Root CA to obtain the service certificate.

The steps involved are shown below.

The **create-csr** command creates a Certificate Signing Request (CSR) file. The arguments to this command are:

- the host name — this is encoded in the subjectAltName extension field of the certificate, and as part of the Subject CN field. The Fully-Qualified Domain Name for the host is required, not its IP address. Some client programs, such as the CDA client, will check this field during the SSL handshake against the FQDN of the server they are attempting to connect to.
- the name of your organisation.
- the name of the application. This is incorporate into the Subject CN field as <fqhn>/<application>. E.g. for a CDA server at host, the Subject field would include `CN=host/cda`.
  Legitimate values for the application argument are:
  - `cda` The Certificate Distribution Authority server
    NB This is normally created with the "create-rootca-cda" command at the same time that the Root CA is created
  - `rca` The Resource Certification Authority server
  - `vops` The VO Policy Service server
  - `mrc` The XtreemFS Metadata and Replica Catalogue Server
  - `dir` The XtreemFS Directory Service
  - `osd` The XtreemFS Object Storage Device server
  - `xtfs_mount` The XtreemFS mount client
  - `volife` The VOLife web application

An example, creating a request for a service certificate, where `host` is replaced with either the Fully-Qualified Domain Name for the host, or its IP address. The last argument to this command identifies the type of service/client that this certificate will be used by (Figure 8).

```
create-csr host "My Organization" cda
```

Figure 8: Creating a request for a CDA service certificate.

The command in Figure 8 produces a private key for the application in `host-cda.key`, and a CSR in `host-cda.csr`. Send this CSR file to the administrator of the Root CA in your organization to get an service certificate (e.g. `host-cda.crt`) in return. Install this service certificate in `/etc/xos/truststore/certs/cda.crt` and the private key in `/etc/xos/truststore/private/cda.key`.

The passphrase protecting the key can be specified in the properties/configuration file of the server it is to be used with. In this case, you must ensure that the file containing the passphrase is only readable by the owner of the service itself, e.g. for the CDA server, the properties file should only be readable by '`cdauser`'.

Connecting the CDA server to the X-VOMS database The CDA server uses the Hibernate ORM library to retrieve VO attributes from the X-VOMS. Hibernate uses a JDBC connection that is specified by the parameters in the Hibernate configuration file, `hibernate.cfg.xml`. The settings that may need to be changed here are `connection.username` and `connection.password`.

3.2 Installation and Configuration of RCA

The Resource Certification Authority services run as DIXI services [6].

RCA comes in two packages:

- **dixi-vom-rca-node** — This package contains the node level service which should run on each node capable of executing jobs. Also the nodes that do not execute jobs, but host services that need to provide certificates (such as the machine VO certificate) require this package to be installed.
- **dixi-vom-rca-server** — This package contains the core-side service which usually runs on one node within a physical organisation.

If needed, the user can also install the command-line tools for working with RCA:

- **dixi-xati** — The package containing the client side libraries and the utilities for accessing the DIXI service functionalities (including the RCA).

To install the necessary software, simply use `urpmi` with the name of the package.

For the normal operation of the RCA client, the node running the RCA client's service also needs to run the AEM's Resource Monitor, which will be automatically installed and configured as a dependency.

3.2.1 Enabling services in DIXI daemon's configuration.

After the package installation, the services will be enabled by default. The service start-up in DIXI is set by the .stage files placed by default in `/etc/xos/config/xosd_stages`. The files for the RCA are as follows:

- **RCA server**: RCAServer.stage
- **RCA client**: RCAClient.stage

The services can be disabled by removing the appropriate .stage file or changing the value of `enabled` to `false` in its contents. Any change in the .stage files requires the xosd to be restarted for the change to take effect.

3.2.2 Configuring core-level RCA service.

The RCA server service creates and uses the **RCAServerConfig.conf** located in standard `/etc/xos/config/` to obtain the configuration:

- **certDNLocation** — the location of the organisation covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **certDNCountry** — the country of the organisation covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **certDNOrganisation** — the name of the organisation covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **certDNOrganisationUnit** — the name of the organisation unit covered by the RCA server. The value is a part of the distinguished name (DN) of a certified resource.
- **daysCertValidity** — The number of days the certificate will be valid, starting from the day of certification and expiring this number of days later.
- **privateKey** — The path to the server's certificate authority's private key.

- **certificateFileName** — The path to the server's certificate authority's public key/certificate.
- **certPassword** — The server's certificate authority's public key's passphrase.
- **keyPassword** — The server's certificate authority's private key's passphrase.
- **rcaDBFile** — The path to the file containing the RCA DB.
- **attributeType** — the type of the attribute certificates. Use V2 for attribute certificates, or V3 for certificates with attributes stored in extensions. The default value is V3, and it is a recommended value for compatiblity with openssl libraries.

The RCA server requires a private key and a certificate signed by a certification authority that is trusted by the nodes in the XtreemOS. The RCA server will use the certificate signed by a commonly trusted root authority to sign the machine certificate requests. The location of the private key and the certificate are defined by **privateKey** and **certificateFileName** of the **RCAServerConfig.conf**, respectively. The RCA will use the password stored in `keyPassword` to decrypt the private key. Similarly, if the public certificate is encrypted, it will use the value of `certPassword`. Usually the `certPassword` will be left blank, however.

The following listing shows an example of an `RCAServerConfig.conf`:

```
# define what will be put in DN of RCA-signed certificates
certDNCountry=SL
certDNOrganisation=XLAB
certDNOrganisationUnit=Research
certDNLocation=Ljubljana

# use the certificate format compatible with openssl
attributeType=V3

# the validity of each certificate signed by RCA
daysCertValidity=30

# RCA service certificate details
certificateFileName=/etc/xos/truststore/certs/rcaserver.crt
certPassword=

# RCA service private key details
privateKey=/etc/xos/truststore/private/rcaserver.key
keyPassword=xtreemos

rcaDBFile=/etc/xos/RCADB.dat
```

3.2.3 Configuring node-level RCA service.

The RCA client service creates and uses the **RCAClientConfig.conf** located in standard `/etc/xos/config/` to obtain the configuration:

- **rcaCertificateFileName** — the path to the RCA server's certificate authority's public key/certificate.
- **resPrivateKeyFileName** — the path to the resource's private key.
- **resIdentityCertFileName** — the path to the resource's identity certificate (public key).
- **resAttributeCertFileName** — the path to the resource's attribute certificate (attribute certificate).
- **resAttributeCertExtFileName** — the path to the resource's attribute certificate (attributes stored in an extension).
- **resVOAttributeCertIncoming** — the path to the folder that will store the attribute certificates pushed from the RCA Server.
- **resKeyPassword** — the passphrase that will be used for encrypting the generated private keys.
- **nonComputationResourceNode** — indicate whether this node will be used as a service node only or not. The value false signifies an ordinary computation resource (i.e., it will accept jobs for execution), while the value true represents a service node that will not be usable for jobs.

```
# the paths where RCA will store and access keys
resPrivateKeyFileName=/etc/xos/truststore/private/resource.key
resAttributeCertExtFileName=/etc/xos/truststore/certs/attrextcert.crt
resAttributeCertFileName=/etc/xos/truststore/certs/attrcert.crt
cdaCertificateFileName=/etc/xos/truststore/certs/rcaserver.crt
resVOAttributeCertIncoming=/etc/xos/truststore/certs/incoming/
resIdentityCertFileName=/etc/xos/truststore/certs/resource.crt

# the key will be encrypted using this passphrase
resKeyPassword=xtreemos

# we want a node to run the jobs
nonComputationResourceNode=false
```

3.3 Installation and Configuration of VOPS

VOPS is a **core-level service** which, due to a usage of the DIXI framework, runs as a service using DIXI communication stages. Please refer to Sections 2.10, 3.9 or [6] for details. VOPS is started in a way like other XOS daemons are started: using *xosd* script provided in a bundle containing a VOPS package. First, administrator has to set up configuration files appropriately, namely **XOSdConfig.conf** and **VOPSConfig.conf**. **ResMng.conf** (on server, where ResMng service is running) must also be configured appropriately to use VOPS as depicted in figure 11. VOPS is a server primarily intended serving requests and forwarding answers from/to resource discovery services and therefore it needs a private key and a public certificate to be able to digitally sign its decisions before forwarding them to the services. The services querying VOPS should have access to the VOPS public certificate to be able to check authenticity of its answers. To obtain the VOPS server key/certificate please refer to Section 3.1.3 where steps for obtaining the server certificate is described (how to use **create-csr** command).

3.3.1 Certificates

VOPS needs the service certificate and a private key for signing its decisions. Please refer to 3.1.3 for instructions on how to create a service certification request for the VOPS. This will also create the private key. Once the administrator processes the certification request and sends the service's public certificate, install the private key and certificate on the server hosting the VOPS:

```
(root)# cp MyHostName-vops.crt \
    /etc/xos/truststore/certs/vops.crt
(root)# mv MyHostName-vops.key \
    /etc/xos/truststore/private/vops.key
```

3.3.2 Installation

To install the necessary software, simply use `urpmi` with the name of the package. Package needed to the VOPS:

dixi-vom-vops —The VOPS service provides means to store and manage VO-level policies, to obtain the policy filters and the policy decisions on the VO level.

3.3.3 Configuration

To be able to run VOPS server using DIXI framework, place a file with a file name **VOPS.stage** into `/etc/xos/config/xosd_stages`. Files content should be similar to this:

```
service=eu.xtreemos.xosd.security.vops.service.VOPSHandler
enabled=true
numThreads=1
```

Figure 9: A sample VOPS stage file.

If configuration file **VOPSConfig.conf** does not exist yet, you can run xosd and stop it:

```
(root)# service xosd start
(root)# service xosd stop
```

This way VOPSConfig.conf is automatically generated under /etc/xos/config, where you can edit it manually (see figure 10).

The content of the configuration file should resemble to this:

```
keyPassword=xtreemos
serviceKey=/etc/xos/truststore/private/vops.key

enableAccessControl=true
VOAdminRoles.size=15
VOAdminRoles.0=vo_administrator
ResourceAdminRoles.size=15
ResourceAdminRoles.0=res_administrator

policyStorage=/usr/share/dixi/VOPS/files/policy/demoStorage
backupStoragePath=/usr/share/dixi/VOPS/storage/backup
```

Figure 10: A sample VOPS configuration file.

- **keyPassword** is a password used to access private service key.
- **serviceKey** is VOPS's private key used to sign responses.
- **enableAccessControl** enables or disables access control: if enabled, extension (role) from user certificate is checked whether it is one from roles listed under **VOAdminRoles** or **ResourceAdminRoles**.
- **VOAdminRoles.size** is the size of array defining VO administrator roles.
- **VOAdminRoles** are roles of users which are permitted to manipulate with XACML policies. These roles must be same as roles specified in certificates (VO administrator roles).

- **ResourceAdminRoles.size** is the size of array defining resource administrator roles.
- **ResourceAdminRoles** are roles of users which are permitted to manipulate with XACML policies. These roles must be same as roles specified in certificates (resource administrator roles).
- entry **policyStorage** points to eXist storage which contains data for storing XMLs.
- entry **backupStoragePath** points to a path to a directory where XMLs are stored when backup or restore commands are issued.

```
VOPSPubCert=/etc/xos/truststore/certs/vopsserver.pem
testVOPS=true
useADS=true
```

Figure 11: A sample ResMng configuration file.

While resource discovery services have to check authenticity of the VOPS's answers, the node running **ResMng** service has to include next lines in its configuration files. List of entries under **ResMng** configuration file:

- **VOPSPubCert** is path to public certificate of the vops server.
- **testVOPS** enables or disables calls to VOPS service.

3.4 Installation and Configuration of Web Front-Ends

Web front-ends run as web applications deployed in Tomcat container. The requirements to install web front-ends include Java Development Kit 1.5 above, Apache Tomcat 5.0 above and MySQL 5.0.45 or above.

Web front-end is implemented in two parts: backend and frontend.

- **backend** — This part contains the VO database setup sql statements, and java library (i.e. JAR files) that interfaces to the VO and RCA related core services by the web interfaces.
- **frontend** — This package forms the core of web application web front-end which exposes the VO and RCA related functionalities for nontechnical users with limited computer expertise.

3.4.1 Installation and Configuration of VO Front-Ends

VO front-end installation comprises of the following steps:

1) Install the X-VOMS database. This step can be skipped if the X-VOMS database has been installed. Please follow "Configuring the X-VOMS database"(3.1.2) section.

2) Deploy the VO frontend part as a web application running in Tomcat container: put the frontend part under directory tomcatdir/webapps/volifecycle (Figure 12).

```
Root # cp volife/frontend /usr/share/tomcat5/webapps/volifecycle
```

Figure 12: Deployment of the VO frontend.

3) Install java libraries for VO web front-end application. The java libraries to install are backend library and their dependent libraries, which should be put into directories WEB-INF/classes and WEB-INF/lib respectively. By default the step can be omitted as the java libraries has been included in the deployable frontend part (Figure 13).

```
# cd volife/backend;
# ant   //build the backend classes,
//assuming the dependent java jar files are under
//volife/frontend/WEB-INF/lib/
# cd ../../
# cp -af volife/backend/build/org
/usr/share/tomcat5/webapps/volifecycle/WEB-INF/classes/.
# cp volife/backend/build/eu
/usr/share/tomcat5/webapps/volifecycle/WEB-INF/classes/.
```

Figure 13: Installation of Java libraries for VO web front-end.

4) Configure the VO web application. One configuration file that may need modification is for sign up notification email.

Configuring sign up notification email. The configuration file can be found under directory tomcatdir/conf/mail.conf. The file configures the email account that sends out sign up notification to a grid user when he has signed up. The format of the configuration file is compatible with JavaMail config file. The common configuration items are given as follows:

- **mail.smtp.host** — the hostname or IP address of the SMTP server the sends out the notitificaiton email.

- **mail.smtp.auth** — the flag that indicates whether the email sending through the smtp server should be authenticated, with two possible values *true* and *false*.

- **mail.smtp.port** — the port number of the smtp server.

- **mail.smtp.user** — the username of the email account, acting as the sender of the notification email.

- **mail.smtp.password** — the password for the email account that authenticates to the smtp server.

An example configuration is presented in Figure 14.

```
mail.smtp.host=smtp.gmail.com
mail.smtp.auth=true
mail.smtp.port=465
mail.smtp.socketFactory.port=465
mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
mail.smtp.socketFactory.fallback=false
mail.smtp.password=PASSWD
mail.smtp.user=MAILACCOUNT
```

Figure 14: Example configuration sign up notification email.

3.4.2 Installation and Configuration of RCA Front-Ends

RCA front-end installation comprises of the following steps:

1) Deploy the RCA frontend part as a web application running in Tomcat container: put the frontend part under directory tomcatdir/webapps/rcalifecycle (Figure 15).

```
Root # cp rcalife/frontend /usr/share/tomcat5/webapps/rcalifecycle
```

Figure 15: Deployment of the RCA frontend.

2) Install java libraries for RCA web front-end application. The java libraries to install are backend library and their dependent libraries, which should be put into directories WEB-INF/classes and WEB-INF/lib respectively. By default the step can be omitted as the java libraries has been included in the deployable frontend part (Figure 16).

```
# cd rcalife/backend;
# ant   //build the backend classes,
//assuming the dependent java jar files are under
//rcalife/frontend/WEB-INF/lib/
# cd ../../
# cp -af rcalife/backend/build/org
/usr/share/tomcat5/webapps/rcalifecycle/WEB-INF/classes/.
# cp rcalife/backend/build/eu
/usr/share/tomcat5/webapps/rcalifecycle/WEB-INF/classes/.
```

Figure 16: Installation of Java libraries for RCA web front-end.

3.5 Installation and Configuration of Monitoring

3.5.1 Installing Monitoring service.

Monitoring service runs as DIXI service. It comes in two packages:

- **dixi-monitoring** — This package contains core monitoring service which should run on every node-level node.
- **dixi-monitoring-dir** — This package contains monitoring directory service which should run on every core-level node.

To install the software, use `urpmi` with the name of the package.

3.5.2 Enabling services in DIXI daemon's configuration.

After the package installation, the services will be enabled by default. The service start-up in DIXI is set by the .stage files placed by default in `/etc/xos/config/xosd_stages`. The files for the Monitoring are as follows:

- **Monitoring**: MonMng.stage
- **Monitoring Directory**: MonDir.stage

The services can be disabled by removing the appropriate .stage file or changing the value of enabled to false in its contents. Any change in the .stage files requires the xosd to be restarted for the change to take effect.

3.5.3 Configuring Monitoring.

Monitoring service creates and uses **MonMng.conf** located in standard `/etc/xos/config/` to obtain the configuration:

- **trustStore** — indicates the path to the folder containing the signer certificates trusted in the AEM or other services.
- **hibernateConfig** — path to the Hibernate configuration file.
- **hibernateMappings** — path to the Hibernate mappings file.

3.5.4 Configuring Monitoring Directory.

Monitoring Directory (currently) does not have any configuration.

3.6 Installation and Configuration of Auditing

3.6.1 Installing Auditing service.

Auditing service runs as DIXI service. It comes in one package:

- **dixi-auditing** — This package contains Auditing service and because Auditing is centalized it should be installed on one node only.

Auditing service provides embedded database for data archiving. If other relational database is intended to be used (e.g. MySQL), then that database should also be installed. Which database is used is specified in the Auditing configuration.

To install the software, use `urpmi` with the name of the package.

3.6.2 Enabling services in DIXI daemon's configuration.

After the package installation, the services will be enabled by default. The service start-up in DIXI is set by the .stage files placed by default in `/etc/xos/config/xosd_stages`. The files for the Monitoring are as follows:

- **Auditing**: AuditingMng.stage

The services can be disabled by removing the appropriate .stage file or changing the value of enabled to false in its contents. Any change in the .stage files requires the xosd to be restarted for the change to take effect.

3.6.3 Configuring Auditing.

Auditing service creates and uses **AuditingMng.conf** located in standard `/etc/xos/config/` to obtain the configuration:

- **trustStore** — indicates the path to the folder containing the signer certificates trusted in the AEM or other services.
- **hibernateConfig** — path to the Hibernate configuration file.
- **hibernateMappings** — path to the Hibernate mappings file.
- **dbPath** — location where the data is stored when embedded database is used.

3.7 Installation and Configuration of Single Sign-On

The `XOS-SSO` system for XtreemOS is delivered in 2 packages: package `XOS-SSO-core` for core (trusted) nodes and package `XOS-SSO-client` for untrusted nodes.

Package `XOS-SSO-core` installs the `SSO-XOS` service, the `libSSO` user library and the `ssh-xos-sso-server` SSH subsystem. The `SSO-XOS` service has a configuration file located in `/etc/xos/config/ssod.cfg`. This configuration file mainly defines the pathnames of the service certificates. The package also installs the start-up scripts in `/etc/init.d`. The `SSO-XOS` service can also be started by users in order to provide private instances of the service. In this case the service reads both the default configuration file `/etc/xos/config/ssod.cfg` and the user configuration file `$HOME/.sso/ssod.cfg`. The configuration files for library routines `libSSO` are located in `/etc/xos/config/sso.cfg` and `$HOME/.xos/sso.cfg`.

Package `XOS-SSO-client` installs the `libuntrustedSSO` library and the `ssh-xos-sso-client` binary which is used by the `libuntrustedSSO` in order to communicate with some `SSO-XOS` service running on a core node. Library `libuntrustedSSO` reads its configuration from `$HOME/.xos/sso.cfg` by default or `/etc/xos/config/sso.cfg` if the user does not provide any configuration. These configuration files allow mainly to configure how to locate potential `SSO-XOS` service.

3.8 Installation and Configuration of Isolation

Functions related to resource isolation require following components which are already included in the standard XtreemOS distribution:

- xos-amsd: the Account Mapping System (AMS) daemon. Use `#service xos-amsd restart` to start the daemon.
- the Linux kernel with cgroup support configured. The kernel should be patched against network flow control support if the administrator wish to enable network flow isolation option.
- Application Execution Management (AEM).

To enable resource isolation, uncomment following lines in the file `/etc/xos/nss_pam/pam_xos.conf`:

```
UseCgroups              yes
Subsystems              cpuacct,memory
```

Available isolation subsystems are:

- cpuacct: CPU accounting control.
- memory: Memory Resource control.
- netlimlit: Network flow control.
- disk: Disk quota limiting.

3.9 Installation and Configuration of DIXI

DIXI installs to the system through the `urpmi` command, requesting the following packages:

- **dixi-main**. The package contains the core classes needed to run DIXI, XATI and the services. This package is required on the nodes that will run any services.
- **dixi-services**. The classes implementing the main support services, needed by other services.
- **dixi-xati**. XATI, the collection of API and client-side access points for running clients. It also contains sample client programs and scripts for running them.
- **dixi-cxati**. Optional installation package, containing additional libraries and programs written and compiled in C.

The dedicated deliverable [6] contains the details on configuring the DIXI. In this document, we will focus only on the security-related specifics of the DIXI configuration.

By default, the `/etc/xos/conf` will contain the `XOSdConfig.conf` with the configuration for the DIXI daemon (the xosd). The following options are relevant to the WP3.5:

- **useSecureXOSd** — sets whether the instance of the xosd should run in the secure xosd mode or not. The secure xosd disables certain features like message redirection, providing a safer environment for the services.
- **secureXOSdConfigLocation** — if `useSecureXOSd` is set to `true`, this option becomes relevant. It contains the path to the the configuration file, which contains the service directory. Please see below for further details.
- **useAccessControlList** — enables or disables the access control for the service calls. Setting the option to `true` will enable the checking of the service messages so that only the authorised users or services will be able to use the restricted methods. The `accessControlListPath` option is related to this option if set to `true`.
- **accessControlListPath** — contains the path to the access control list, used by the xosd if the `useAccessControlList` is set to `true`. Please see below for further details.
- **useSSL** — indicates whether the communication should use SSL for security. Default value: *false*.
- **trustStoreSSL** — indicates the path to the folder containing the signer certificates trusted in the SSL handshakes.
- **trustStore** — indicates the path to the folder containing the signer certificates trusted in the AEM or other services. If **useAccessControlList** is enabled, the certificates in this path will also provide information on which credentials can be trusted.
- **certificateLocation** — the path to the public certificate used for the SSL handshakes. The certificate needs to be able to handle both server and client connections.
- **privateKeyLocation** — the path to the private key used for the SSL handshakes and for encrypting the communication.

To apply changes to the configuration file, the xosd needs to be restarted:

```
# sudo service xosd restart
```

Access control lists. The access control lists are configuration files which contain the following information:

- The list of valid categories. Multiple service call methods can belong to each category.
- A list of groups and roles assigned to each category. This list represents who can use the methods of the category. If the list contains the asterisk ($\star$), anyone is permitted to invoke the service calls of the category.

The default access control list installs as `acl.conf` in the /etc/xos/config. The following is an example of an access control list:

```
registration_info=*
resource_administration=admins:resource_administrator, admins:administrator
site_administration=admins:site_administrator, admins:administrator
vo_administration=admins:vo_administrator, admins:administrator
administration=admins:administrator
```

Static service directory. If the `useSecureXOSd` option is enabled in the configuration, the `secureXOSdConfigLocation` needs to point to the local service directory. By default this is stored in the `secure_xosd.conf` stored in the standard xos configuration path. The file contains a simple list of `service=address` pairs, e.g.,

```
eu.xtreemos.xosd.security.vops.service.VOPSHandler=172.16.93.104:60000
eu.xtreemos.xosd.resmng.ResMng=172.16.93.104:64000
eu.xtreemos.xosd.security.rca.client.service.RCAServerHandler:172.118.16.40:60000
```

3.9.1 Installing the first XtreemOS node

When setting up XtreemOS for the first time, we recommend the following steps, performed by the resource administrator on the node using the root account:

- Designate a node that will be the first core node.
- Install and configure both the core-level and the node-level RCA service (3.2). Please make sure the node has the proper RCA's certificate and key files in place.
- Optionally, for security, set the node's firewalls to block any incoming traffic.

- Disable SSL by setting **useSSL** to **false** in **XOSdConfig.conf**.
- Disable SSL for the clients by setting **useSSL** to **false** in **/root/.xos/XATIConfig.conf**.
- Start the **xosd** service by calling `service xosd start`.
- Obtain the machine identity certificate by calling, in sequence: `rca_apply`, `rca_confirm`, `rca_request`
- Stop the **xosd** service by calling `service xosd stop`.
- Restore any firewalls to permit normal incoming traffic.
- Re-enable SSL by setting **useSSL** back to **true** in **XOSdConfig.conf**. Set the password to the **privateKeyPassword** value. Other settings should already point to the machine certificate and key by default.
- Re-enable SSL for the command-line administration commands by setting **useSSL** to **true** in **/root/.xos/XATIConfig.conf**. Make sure that **sslPrivateKeyPassword** contains the correct password.
- Obtain the resource administrator's user certificate and install it to the default folders.
- Next time the **xosd** service will start, it will operate using SSL only.

3.9.2 Adding a new node to a pre-existing XtreemOS site

When adding a new node to the network running XtreemOS with SSL, it is possible to use of the resource administrator's user certificate for boot-strapping the SSL for the new node. We recommend the following steps to be performed by the new resource's administrator using the root account:

- Obtain (using **get-xos-cert**) or install the administrator's user certificate and key.
- Open **/root/.xos/XATIConfig.conf** and make sure that the **sslPrivateKeyPassword** value contains the proper value.
- Open **XOSdConfig.conf** for text editing.
- Insert the hash (#) character to the beginning of the lines containing **privateKeyLocation**, **certificateLocation** and **privateKeyPassword** to comment them out.

- Add a new entry for **certificateLocation**, using the administrator's user certificate as the value (e.g., `/root/.xos/truststore/cert/user.crt`).
- Add a new entry for **privateKeyLocation**, using the administrator's private key as the value (e.g., `/root/.xos/truststore/private/user.key`).
- Add a new entry for **privateKeyPassword**, using the password to the administrator's private key as the value.
- Save the changes to **XOSdConfig.conf**.
- Install and configure the node-level RCA service (3.2). Please make sure that the RCA server's certificate is installed.
- Start the **xosd** service by calling `service xosd start`.
- Apply for the resource registration using `rca_apply`.
- The site administrator then needs to confirm the resource entry using `rca_confirm`.
- Obtain the node's machine identity certificate by calling `rca_request`.
- Stop the **xosd** service by calling `service xosd stop`.
- Open **XOSdConfig.conf** for text editing.
- Comment out using the hash (#) character the **privateKeyLocation**, **certificateLocation** and **privateKeyPassword** that are set to the administrator's user certificate and private key.
- Remove the hash (#) character from beginning of the lines with the original values for **privateKeyLocation**, **certificateLocation** and **privateKeyPassword**.
- Save the changes to **XOSdConfig.conf**.
- Use the xosd normally.

4 User Guide for Security Services

4.1 Using X-VOMS

4.1.1 Introduction

The XtreemOS user can access the following features in X-VOMS:

- Identity Management - Store details of user identity, secure their access to system
- VO Management - Allow creating/joining VOs
- Credential Management - generate private keys and XOS certificate containing user's public key and VO attributes, also generate private key and public key certificates for core services

The interface to these features is available via web front-ends and is described in Section **??**. In addtion, the CDA client command-line tool provides a scriptable interface to the CDA server.

4.1.2 CDA Client

The CDA client can be used to generate a user's private key and obtain an XOS Certificate containing their public key and VO attributes. Optionally, the CDA client can use an existing private key file or an existing Certificate Signing Request file, either of which can be saved from a previous invocation of the client. Reusing an existing private key or CSR file can be more convenient than generating a new private key and setting a new pass-phrase.

The CDA client allows the user to request a duration for the validity of their credentials. All parameters may be entered as arguments on the command-line, and pass-phrases may be read from files, allowing them to be passed securely to the command (that is, without showing in the output from "ps").

Requesting a user XOS certificate with get-xos-cert The CDA client is invoked by the command `get-xos-cert`. The example in Figure 17 shows how to request a certificate for the user `xtreemos-vouser` belonging to the VO `exampleVO` and group `group1`.

```
$ get-xos-cert host:6730 exampleVO group1
Enter your username: xtreemos-vouser
Enter password: <not echoed>

Passphrase to protect private key
(at least 8 characters long): not echoed
Type passphrase again to confirm: not echoed

saving private key file in
/home/user/.xos/truststore/private/user.key
saving certificate chain (user+CDA) in
/home/user/.xos/truststore/certs/user.crt
```

Figure 17: Example user requesting certificate.

The example in Figure 17 retrieves the user's VO attributes for the VO *vo1* and primary group *group1*. The username supplied is that of the pre-defined user *xtreemos-vouser* (password *xtreemos*). The command saves the certificate chain consisting of the XOS-Certificate and the issuer's certificate, namely the CDA service certificate.

The values of the user's VO attributes can be viewed with the command *view-xos-cert*, as shown in Figure 18.

```
$ view-xos-cert /home/user/.xos/truststore/certs/user.crt
Certificate:
   Data:
       Version: 3 (0x2)
       Serial Number:
           01:21:43:39:3c:16
       Signature Algorithm: sha256WithRSAEncryption
       Issuer: O=XtreemOS, OU=cda, CN=host/cda
       Validity
           Not Before: May 15 07:37:53 2010 GMT
           Not After : Jun  4 07:47:53 2011 GMT
       Subject: CN=ea9a7366-e34f-4a99-9e31-277430366475
       X509v3 extensions:
           X509v3 Basic Constraints: critical
               CA:FALSE
           X509v3 Key Usage: critical
               Digital Signature, Key Encipherment
           X509v3 Extended Key Usage: critical
               TLS Web Client Authentication
... (Details excluded)

       XtreemOS VO Attributes:

           GlobalPrimaryVOName:
               2c0e8cb2-4453-46fe-85b7-74874e76e7c2
           GlobalSecondaryVONames:
               null
           GlobalUserID:
               ea9a7366-e34f-4a99-9e31-277430366475
           GlobalPrimaryGroupName:
               ae88816f-9f5c-48f9-ad7d-f71a64977904
           GlobalSecondaryGroupNames:\
41ef52b1-8c03-4305-bbfb-9f07245702cd,\
9d481237-26c9-4854-a1e2-e23d1a61059c
           Role:
               null
           Group:
               group1
           Subgroup:
               null
```

Figure 18: Viewing a XOS certificate.

Requesting a service certificate with get-service-cert An administrator can request a service certificate with the following command:

```
$ get-service-cert host:6730 hostname type 'Short description '
Enter your username: xtreemos-admin
Enter password: <not echoed>

Passphrase to protect private key
(at least 8 characters long): not echoed
Type passphrase again to confirm: not echoed

saving private key file in hostname-type.key
saving certificate chain (type+CDA) in hostname-type.crt
```

Figure 19: Example requesting a service certificate.

The private key and service certificate then needs to be placed in the /etc/x-os/truststore/private,certs as appropriate and the properties file for the core service modified to include these file locations and the pass-phrase protecting the private key.

Verifying the certificate received by CDA client Before saving the certificate to disk, the CDA client verifies the certificate by checking that it has been signed by the CDA server, and that the CDA service certificate received during the SSL handshake has been signed by the Root CA. You can verify the certificate at any other time with the command "verify-cert" as follows:

```
$ verify-cert user.crt
user.crt: OK
```

Figure 20: Verifying a certificate

This command is a wrapper around "openssl verify" which uses the issuer certificate stored by the CDA client along with the user certificate to verify the certificate chain. If there is an error, such as a signature mismatch, the command reports "FAILED", and sets the exit status to "1" (to enable a shell script to test the certificate status). This is also the case if the user certificate has expired. Optionally, if a Certificate Revocation List (CRL) is available, this can be specified as a second argument, and then the command will check whether the certificate has been revoked.

4.1.3 X-VOMS Guide for VO Administrators

Revoking User Certificates The Grid administrator can revoke user XOS-Certificates by specifying the serial number(s) to the "revoke-xos-cert" command:

```
# revoke-xos-cert  serial1 serial2 serial3
```

Figure 21: Revoking a XOS-Certificate(s) in the X-VOMS database.

(Here, the backslash "\" has its conventional meaning of continuing the command text over a newline.)

The command needs to be run with root permissions or the privileges of the "cdauser" user, to check that the user is authorised to use the command. If the certificate(s) specified have not expired yet, this command will set the the certificate status to "revoked" in the X-VOMS database. The arguments to the command are a list of certificate serial numbers that are to be revoked. (NB "openssl x509" prints the certificate serial number with colons ":" separating each pair of hex digits; Java's "keytool -print-cert" command prints the serial number in a more useful format.)

The "revoke-xos-cert" command can be run either as soon as a user notifies they have lost their private key, or at other intervals, as determined by the administrators of this XtreemOS Grid.

The command to produce a CRL from the status information in the X-VOMS database is "create-crl", as shown below:

```
# create-crl
Creating CRL 12763444101 for certifcate(s) with serial
number(s) serial1 serial2 serial3
```

Figure 22: Creating a Certificate Revocation List from status information in the X-VOMS database.

This CRL file can now be made available to nodes which need to check the revocation status of XOS-Certificates. The frequency at which the CRL file is generated and distributed will vary according to the needs of a particular XtreemOS Grid. These two commands are separate to enable more flexibility in the creation of CRLs.

One method of publishishing the CRL is to place it on the front page of the VO-Life web application, to enable administrators of resource nodes to download it as

required. Another mechanism being investigated is the use of the Scalaris publish-subscribe mechanism to push the CRL to all resource nodes in an XtreemOS Grid, ensuring a more timely notification of revoked certificates.

Using the CRL to check a certificate's revocation status The CRL file can be used with the "check-cert-revocation" command (NB Not implemented yet).

Operating the CDA Manually The following method of creating a service certificate is deprecated. Instead, service certificates should be generated with the command "get-service-cert".

Manual method of creating service certificates with the CDA private key This manual method consists of using the CDA to sign Certificate Signing Requests for core services. The manual operating mode of the XtreemOS CDA involves taking Certificate Signing Requests (CSR files) for core services and convert them to service certificates. This function requires access to the directory holding the Root CA, but only requires access to the CDA private key and public key certificate (the Root CA private key is not involved).

The CSR file can be sent to the operator of the CDA in an email message or by other means. The operator of the CDA should save the CSR file and examine it to check its validity, and contact the originator of the request if necessary (to verify its source).

For example, figure 23 shows the contents of a request for the VOPS server on the host `host` (the values in your CSR files will, of course, be different).

```
$ view-csr host-vops.csr
Certificate Request:
 Data:
  Version: 0 (0x0)
  Subject: CN=host/vops, O=XtreemOS Project,OU=vops
  Subject Public Key Info:
   Public Key Algorithm: rsaEncryption
   RSA Public Key: (1024 bit)
    Modulus (1024 bit):
     00:d0:ec:ed:89:93:2e:c3:23:47:7c:30:1e:de:fb:
     40:bb:9a:6f:bd:77:35:54:28:24:b2:62:9b:cc:9e:
     dd:f5:1b:19:55:be:fe:0b:9f:2d:56:a6:98:bd:77:
     53:1c:da:38:3a:ba:60:03:90:f9:bc:a4:af:ec:5a:
     c1:ec:80:34:cb:bd:fa:18:93:af:c1:84:5d:16:72:
     ed:94:e8:eb:59:13:1f:99:6b:ac:93:d3:e5:07:e1:
     54:77:e8:8f:44:4c:4a:0b:31:5c:26:af:19:c3:f6:
     6c:71:22:cb:0c:56:47:99:f3:14:ab:1b:43:de:e9:
     13:48:17:00:f0:da:0c:de:e1
    Exponent: 65537 (0x10001)
   Attributes:
   Requested Extensions:
    X509v3 Basic Constraints:
     CA:TRUE
    X509v3 Subject Alternative Name:
     DNS:host
    Signature Algorithm: sha1WithRSAEncryption
      5a:c2:4e:aa:8f:bc:e2:c5:b2:0a:12:20:92:d5:90:de:fb:96:
      bb:d7:c3:5f:6d:67:89:5e:b1:6c:e1:9e:c6:e7:8e:e9:42:ea:
      0e:65:f1:3d:e9:73:31:44:95:6c:d3:3c:b4:b1:cc:bb:e6:1c:
      3c:a2:c1:99:a7:2e:d4:24:10:06:49:99:51:94:53:81:d9:8e:
      4d:a2:f5:1c:ac:df:19:e0:f4:bb:96:19:5f:74:88:57:e3:82:
      01:e7:93:a1:45:cc:3e:ef:54:28:bb:8a:1e:c0:3a:a9:dd:85:
      00:2f:ac:c7:b8:5c:c8:94:99:2f:a7:04:76:d4:74:84:f4:5d:
      a7:92
```

Figure 23: Examining Certificate Signing Request for a service certificate.

Note that the Subject Alternative Name extension field contains the FQHN (Fully-Qualified Host Name) of the machine that this certificate will be used on. This hostname, along with the name of the application, is also encoded in the Subject CN field.

If you trust the originator of this request, and you have validated the request by examining it as shown above, you are now in a position to generate an service

certificate from the request. Figure 24 shows the step needed to create an service certificate from a CSR file, in this case for the VOPS server at `host`.

```
# process-csr /opt/xtreemosca host-vops.csr
```

Figure 24: Signing a core service CSR file, creating an core service certificate.

The service certificate is created in the file `host-vops.crt` in the current working directory. A copy of the service certificate is also stored in the 'certs' sub-directory of the root CA directory. In the example above, this would be `/opt/xtreemosca/certs`), named `<NN>.pem`. A record of the newly issued service certificate is made in `/opt/xtreemosca/index.txt`, including the certificate's expiry date, revocation status, serial number and CN.

Note again, using the CDA to sign certificates from the Root CA directory does not require the Root CA private key to be available - the Root CA private key can be removed from the Root CA directory.

4.2 User Guide for RCA

This section provides the information on using the RCA through the command-line tools. These tools provide access to all RCA's functionality, but many of them are restricted to particular types of administrators.

4.2.1 VO's RCA management

Each RCA can belong to any number of the VOs. When an RCA belongs to the VO, the resources registered with the RCA can provide services to this VO. The `rca_vo.sh` command provides the manipulation of the VO membership for the RCA:

- **rca_vo a** *VO_id* — add the RCA to the VO with the ID *VO_id*.
- **rca_vo l** — list the VO IDs of the VOs the RCA is currently a member of.
- **rca_vo r** *VO_id* — remove the RCA from the VO. This operation also removes all resources registered in the RCA to belong to the *VO_id* from the VO.

4.2.2 RCA's resource registration management

The RCA contains the list of resources currently registered and those pending to be registered. The site administrator can use the following commands to examine

the currently pending resources and the ones already registered. The pending requests can be confirmed, making the resource fully registered and able to have its machine certificates signed:

- **rca_list_pending** — Display the list of nodes applied for the registration with the RCA. These nodes have been provided by the resource administrator, but the site adminstrator should use a formal off-line process to check the validity of the applicant resource administrator and the resource itself.

- **rca_list_registered** — Display the list of nodes registered with the RCA. These resources are capable of having their resources signed by the RCA.

- **rca_confirm** *Resource_Address* — Confirm the registration of the node in RCA. The *Resource_Address* needs to be of the form *IP*:*Port*, and the values of *IP*:*Port* can be found on the **rca_list_pending** output list. If the resource to be confirmed is the same as the resource used by the administrator, the address parameter can be omitted.

4.2.3 RCA client

The RCA client helps manage the machine certificates (i.e., machine's identity certificate, its attribute certificate and all the VO-related attribute certificates). It generates the private key for the machine certificates and has it signed by the RCA server. It also collects the information on the current node from AEM's Resource Monitor to make the process of the resource registration simpler.

For the machine management enabled by the RCA, the following commands can be used:

- **rca_local_info** — The RCA client holds the information on the current node. This command lets the user choose the manner of the obtaining of the node's information. Primarily, the information can arrive from AEM's Resevation Manager. This information can be stored into /etc/xos/config/ResourceDescriptor.xml and modified in a text editor, then used in the future in a modified form.

- **rca_apply** — Apply for registration of this node with RCA.

- **rca_request** — Request a new resource certificate pair. The script has the RCA client create a new private key, sends the corresponding public key for signing to RCA server and, if properly registered with the RCA, lists the obtained certificate contents. It creates and replaces the files containing the machine certificate and machine's private key into the files configured in

RCAClientConfig.conf as **resIdentityCertFileName** and **resPrivateKeyFileName**, respectively. It also stores the machine's attribute certificate into file defined as **resAttributeCertExtFileName** (unless the RCA server returns a V2 attribute certificate; in this case the file is saved into the path and filename defined by **resAttributeCertFileName**).

- **rca_update** — Collect the most up-to-date information on the resource and update the resource record at the RCA.
- **rca_unregister** or **rca_remove** — Unregister the resource from the RCA.
- **rca_info** — Display the current resource certificate pair details.

4.2.4 VO's resource management

The resource can take part in any number of VOs with the limitation that the RCA has to belong to these VOs in the first place. The following commands can be used by the VO administrators and the resource administrators.

- **rca_resource_vo a** *VO_id* **[***Resource_Address***]** — set the node denoted with *Resource_Address* in the form of *IP*:*Port* to be a member of the VO with *VO_id* for its id. If the *Resource_Address* parameter is omitted, the current node will be added. If the target node is online, the result of the script invocation will be a new VO-related attribute certificate placed into the node's folder defined in **RCAClientConfig.conf** as **resVOAttributeCertIncoming**. To install it, simply move it to the same folder that contains the machine identity certificate.
- **rca_resource_vo r** *VO_id* — remove the current node from a membership in the VO with *VO_id* for its id.
- **rca_resource_vo c** *VO_id* — request a VO-related attribute certificate of VO with *VO_id* for its id for the current node. The result of the script invocation will be an installed new VO-related attribute certificate.

4.3 User Guide for VOPS

This section provides the information on using the VOPS through the commandline tools. VO Policy Service provides a way for users and administrators to manage their policies. Moreover, it enables other DIXI services to enforce access control to resources based on policies residing in VOPS policy storage. VOPS management commands are accessible through `vops.sh` script.

4.3.1 User commands

Users with appropriate credentials can issue several VOPS commands: VO administrators have besides commands for manipulation with VO level policies also access to VOPS management commands, meanwhile resource administrators and VO users have access to VOPS standard commands for manipulation of policies.

User and resource administrator commands available:

- **list ids of policies**
- **list specific policy**
- **add policy**
- **remove policy**

Using these commands, users can list policies, obtain specific policy, add new policy and remove specific policy, respectively. VO administrators have as well access to the following two commands:

- **backup the database**
- **reload the database**

By issuing **backup** command, VO administrator stores all policies residing in VOPS storage to a directory as set in the VOPS configuration file under the **backupStoragePath** variable (see section 3.3). Policies are stored as XML files, more precisely in XACML format. Filenames of these files are the same as policy ids — each policy is stored in XML file with the same name as policy id. With **reload** command current VOPS storage is emptied and policies are reloaded from the backup storage path, so administrator issuing this command must see that policies must first be backed up in order not to loose them.

List policies Administrators and users (policy owners) can list policies they are allowed to edit. This command lists ids of policies of the owner. By knowing the specific id of the policy, the owner can list the policy by issuing the **list policy** command. VOPS has its own access control over the type of user specified by the **domain argument**. VO administrators can view policies which belong to specific VO's, given by the third argument, namely the **complementary id**. VO users provide their GUID and resource administrators provide the id of the resource as the **complementary id** argument.

Command:

```
vops l <domain> <complementary id>
```

Examples:

- `vops l usr ea9a7366-e34f-4a99-9e31-277430266475`. User lists his or her policies residing in VOPS storage.
- `vops l vo eaabg366-e3ff-4a9f-9ef1-27fa30b664g5`. VO administrator lists policies belonging to a VO with provided id.

List policy With this command users and administrators can list a policy with the id returned by command for listing policies (**list** command). Complementary id is user's GUID or id of the VO adminstrator is managing.
Command:
`vops lp <domain> <policy id> <complementary id>`
Example: `vops l usr ea9a7366-e34f-4a99-9e31-277430266475`

Add policies With this command users and administrators add policies. First argument **domain** designates the domain issuer is part of. Second argument presents path to the policy residing locally which will be added into the storage. Argument **complementary id** designates either global user id in case the issuer is user, VO id in case the issuer is VO administrator, or resource id in case the issuer is resource administrator. Last optional argument, **process monitoring** designates the processing of the monitoring rules. If this argument is set to **true**, processing of the monitoring rules is activated. Default value is **false**.

Command: `vops add <domain> <local path to policy> / <complementary id> {<process monitoring>}`
Example: `vops add usr /home/user/mypolicy.xml / ea9a7366-e34f-4a99-9e31-277430266475`

Remove policy Policies can be removed by providing **policy id**, **complementary id** and optionally boolean value for processing monitoring rules from policy, namely **process monitoring**. If last argument is not provided, **false** is default.
Command:
`vops rp <domain> <policy id> <complementary id> {<process monitoring>}`
Example: `vops add usr /home/user/mypolicy.xml / ea9a7366-e34f-4a99-9e31-277430266475`

Backup the database Issued when VO administrator wants to back-up the VOPS policy storage to a path specified as in VOPS configuration file under **backupStoragePath** variable (see section 3.3).

Command: vops b <domain>
Example: `vops b vo`

Reload the database Issued when VO administrator wishes to reload the VOPS policy storage on a server from a location specified in VOPS configuration file under **backupStoragePath** variable (see section 3.3).

Command: vops reload <domain>
Example: `vops reload vo`

PDP consists of methods of the VOPS framework which are used by invoking services requesting control access over a query for e.g. user accessing some resource or user trying to submit a job on a particular resource. These PDP methods of the VOPS are used internally by AEM through Resource Manager service.

4.3.2 Policies

By default VOPS provides a default policy which permits any action of a user to any resource. This default policy is listed in figure 25. Other policies can be added using PAP and commands listed above.

```
<Policy PolicyId="Policy01" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:deny-overrides">
  <Description>
        This is the default policy - it permits access for all users to all
        resources.
  </Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
<Rule RuleId="DefaultRule" Effect="Permit">
    <Description>This rule permits access.</Description>
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
  </Rule>
</Policy>
```

Figure 25: A default XACML policy. This policy permits all actions to all users on a resource in a query.

Non-default policies are listed below. Filter policy is a policy filtered through VO and user policy storage residing in VOPS. An example of a filter policy is listed as figure 26. Its target part conforms to any request. It contains one rule from a VO policy permitting actions on a resource within a specific VO.

```
<Policy PolicyId="FilterPolicy"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
  <Description>Filter policy generated by VOPS.</Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <Rule RuleId="Permit_user_to_abc15788-976a-4b8b-a849-3e6621c3aff5" Effect="Permit">
    <Description>1.policy, 1.rule
     This rule permits all actions on resources inside VO:abc15788-976a-4b8b-a849-3e6621c3aff5.
    </Description>
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <Resource>
          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
              abc15788-976a-4b8b-a849-3e6621c3aff5
            </AttributeValue>
            <ResourceAttributeDesignator AttributeId="resource:extensions:VO"
              DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
  </Rule>
  <Rule RuleId="DefaultRule" Effect="Deny">
    <Description>1.policy, 2.rule</Description>
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
  </Rule>
</Policy>
```

Figure 26: An example of a filter policy.

A resource policy listed as figure 27. It defines a rule permitting a specific user group to execute jobs requiring at most 2 CPUs. All other requests to that resource are going to be denied as default rule denies them.

4.4 User Guide for VO Web Front-End

This section provides the information on using the VO web front-end. The functionalities provided by the VO web front-end include user registration, registered user identity management, VO management. These functionalities provided to different role of users are given as follows.

```
<Policy PolicyId="ResourcePolicy01"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
<Description>
  This policy applies to a resource Address = [://194.249.173.88/194.249.173.88:60000(/194.249.173.88)]
  This policy applies to users from VOgroup1.
  Permits if user wants to have IndividualCPUCount <= 2.
  Permits if user does not explicitly define IndividualCPUCount.
</Description>
 <Target>
  <Subjects>
   <AnySubject/>
  </Subjects>
  <Resources>
   <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
       Address = [://194.249.173.88/194.249.173.88:60000(/194.249.173.88)]
     </AttributeValue>
     <ResourceAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
       DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </ResourceMatch>
   </Resource>
  </Resources>
  <Actions>
   <AnyAction/>
  </Actions>
 </Target>
<Rule RuleId="ResourceRule01VOgroup1" Effect="Permit">
 <Description>
  Enable access if IndividualCPUCount >= 2 for VOgroup1.
 </Description>
<Target>
<Subjects>
 <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">VOgroup1</AttributeValue>
      <SubjectAttributeDesignator AttributeId="subject:extensions:group"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
     </SubjectMatch>
    </Subject>
   </Subjects>
  <Resources>
  <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:double-equal">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#double">2.0</AttributeValue>
    <ResourceAttributeDesignator
      AttributeId="resource:jsdl:JobDefinition:JobDescription:Resources:IndividualCPUCount:UpperBoundedRange"
      DataType="http://www.w3.org/2001/XMLSchema#double" />
   </ResourceMatch>
  </Resource>
 </Resources>
 <Actions>
  <Action>
   <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">action:AEM:SubmitJob</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>
</Rule>
<Rule RuleId="DefaultRule" Effect="Deny">
    <Description>This rule denies access.</Description>
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <AnyAction/>
      </Actions>
    </Target>
  </Rule>
</Policy>
```

Figure 27: An example of a resource policy.

4.4.1 Prospective User

- **Sign up to grid** — Generates user registration to the grid administrator.

4.4.2 Registered Grid User

- **Change password** — Changes the password for current grid user.
- **Generate new keypair** — Generates a PKI key pair and downloads the private key.
- **Get an XOS-Cert** — Obtains a certificate signed by a VO owner.
- **Create a VO** — Creates a user's own VO.
- **Request to join a VO** — Send requests to join others'VOs.
- **Leave a VO** — Leave others'VOs.

4.4.3 The Grid Administrator

- **Approve user registration** — Let a prospective user become a registered grid user.
- **Remove registered user from the grid**

4.4.4 The VO Administrator

- **Manage VOs** — Lists, creates and deletes the current user's owned VOs.
- **Approve user request to join a VO** — Lists, approves and declines the pending requests for joining the VO.
- **Remove user from a VO**

4.5 User Guide for RCA Web Front-End

This section provides the information on using the RCA web front-end. The functionalities provided by the RCA web front-end include register and unregister resource from a RCA, add/remove a RCA to/from a VO, add or remove resource to/from a VO, list the information of the relationship among resource, RCA and VO. These functionalities provided to different role of users are given as follows.

4.5.1 The VO Administrator

- **Add a RCA to the VO**
- **Remove a RCA from the VO**
- **Add resource to the VO that the RCA joined in**
- **Remove Resource from the VO that the RCA joined in**

4.5.2 The Site Administrator

- **Request to add a RCA to a VO**
- **Request to remove a RCA from a VO**
- **List RCA relationship** — List information of the RCA the VOs that the RCA joined.
- **List resource infomation** — Display the list of nodes registered with the RCA.

4.5.3 The Resource Administrator

- **Register resource** — Approve the registration of this node with RCA.
- **Unregister resource** — Unregister the resource from the RCA.
- **Add resource to the VO that the RCA joined in**
- **Remove Resource from the VO that the RCA joined in**

4.6 User Guide for Monitoring

This section provides the information on using Monitoring through the commandline tools. These tools provide means to menage monitoring rules and subscriptions.

4.6.1 User commands

Commands available are:

- **addnotification** *monRule monRuleName* — adds a new monitoring rule defined by *monRule* and associated by *monRuleName*.

- **cancelnotification** *monRuleName* — cancels notification associated by *monRuleName*.
- **existsmonrulename** *monRuleName* — checks if monitoring rule already exists.
- **subscribe** *monRuleName* — subscribes to monitoring rule named *monRuleName*.
- **list rules** — lists all monitoring rules.

4.6.2 Usage examples

Monitoring rule which monitors user level CPU usage of every resource in specific VO is added in the following manner:

```
xmon addnotificaton @{vo:[8a3ac8e0]}cpu_user MyVO-CPU
```

Issuing `xmon list rules` will output:

```
MyVO-CPU: @{vo:[8a3ac8e0]}cpu_user
```

showing we added monitoring rule named `MyVO-CPU`.

We can check if monitoring rule by the name `MyVO-CPU` exists

```
xmon existsmonrulename MyVO-CPU
```

which in our case returns `YES`.

Every time Monitoring service will receive CPU status update of any resource, a callback will be triggered notifying CPU usage status. We can subscribe to these notificatoions using `subscribe` command:

```
xmon subscribe MyVO-CPU
```

Until the execution if this command is canceled (e.g. `Ctrl-C`), all notifications are received:

```
1. Wed Apr 21 11:18:30 CEST 2010,cpu_system,12.7,PERCENT,null,
null,null,null,ResourceID = [IP=172.16.117.205:60000]
2. Wed Apr 21 11:18:36 CEST 2010,cpu_system,12.7,PERCENT,null,
null,null,null,ResourceID = [IP=172.16.117.205:60000]
3. Wed Apr 21 11:18:42 CEST 2010,cpu_system,12.7,PERCENT,null,
null,null,null,ResourceID = [IP=172.16.117.205:60000]
```

More information about monitoring rules can be found in D3.5.13 deliverable.

4.7 User Guide for Auditing

This section provides the information on using Auditing through the commandline tools. These tools provide means to menage archiving rules and to query Auditing database.

4.7.1 User commands

Commands available are:

- **addarchiverule** *monRule* — adds archive rule defined by *monRule*.
- **cancelarchiverule** *archiveRuleId* — cancels archive rule associated by *archiveRuleId*.
- **list rules** — lists all archiving rules.
- **query** *queryString* — performs query with provided *queryString* on Auditing history database.

4.7.2 Usage examples

New archiving rule for archiving user level CPU usage of all resources is added in the following manner:

```
xaudit addarchiverule @cpu_user
```

The output of the latter command is ID of the archiving rule:

```
Archive rule ID: audit_5e3e92f9-d89a-4576-a704-7dede8203791
```

Archiving rules are listed using `list rules` arguments of the `xaudit` command:

```
xaudit list rules
```

The output of the latter command is a list of archiving rule IDs with corresponding archiving rules:

```
audit_8ac72d81-2fb8-458a-aaee-5910ce092c0c: @disk_free
audit_86b73a82-fa08-4752-bbac-c6dd6b5263b5: @job_started
audit_849ec23c-aab9-4f2b-b70c-e17b5b09e794: @mem_free
audit_ab78c3ae-8a2b-4044-8efe-ae15ac70a63d: @cpu_system
audit_7747e3ff-66f4-4de0-877b-34b781b880bf: @mem_total
audit_dc747b66-6738-4fcc-a03f-3add480c6ea7: @cpu_user
audit_e2d6b09a-3e30-4aaa-8da6-ab28ec74712e: @disk_total
audit_fc6a7d47-72ea-436a-a4a1-23d7276b2471: @job_failed
audit_ba4f24c8-b587-4a66-b261-10483143e220: @job_finished
```

To query Auditing database `query` argument of the `xaudit` command must be used followed by Hibernate Query Language (HQL) query string. Below is an example of query string that returns an average of CPU usage where all resources are taken into account:

```
1. xaudit query "select avg(value)
   From FloatMetric where metricName='cpu_user'"
```

More information about archiving rules and Hibernate Query Language can be found in D3.5.13 deliverable.

4.8 User Guide for Single Sign-On

The single sign-on system in XtreemOS is meant to be as transparent as possible for users. Libraries `libSSO` and `libuntrustedSSO` provide the same API as the default `libXATICA` for accessing grid services in XtreemOS. All binaries bound to these new libraries in place of `libXATICA` automatically exploit the single sign-on system.

In complement to `libXATICA`, these libraries provide new entry points to manage user credentials: loading new certificates (`uploadCredential`), renewing certificates (`renewCredential`), deleting them, etc. When the user loads a certificate, the service applies the associated validation procedures immediately (private key challenge and chain validation for X.509 certificates for instance).

In order to maintain compatibility with previously developed codes, the `XOS-SSO` service handles grid requests with a credential parameter in the following way:

- if the credential parameter is empty, the `XOS-SSO` service uses the currently selected credentials for this request. As the selected credentials have already been validated, the `XOS-SSO` does not apply the full validation process. For instance key challenge is avoided in order to provide single sign-on. On the other hand, some tests such as revocation check are applied each time the credentials are used.

- if some credentials are associated to the request, the `XOS-SSO` service first checks if it has already validated these credentials. In order to do so, it keeps all already validated credentials in its memory. In the case where the credentials are unknown to the service, it applies the full validation procedure and, in case of success, transmits the request along with the credentials and keeps the credentials in its memory. In the case where it has already validated these credentials, only a partial validation process is applied (no key challenge).

This behaviour allows to use the single sign-on service without having to explicitely load credentials.

5 Conclusion

This document has presented the third version of the XtreemOS security and VO management services. The services to be included in XtreemOS 3.0 are:

- X-VOMS, XtreemOS VO Management Service
- RCA, XtreemOS Resource Certification Authority
- VOPS, XtreemOS VO Policy Service
- Monitoring service
- Auditing service
- Single Sing-on functionality
- Isolation functionality
- Secure DIXI, XtreemOS communication bus
- Web Front-end to XtreemOS security services

For each service and functionality, this document includes a brief description of its main functionalities, the procedure for installation and configuration, and a user guide.

All these services and functionalities will be included in the release 3.0 of XtreemOS.

References

[1] XtreemOS Consortium. First specification of security services. http://bit.ly/XtreemOS-D353, May 2007.

[2] XtreemOS Consortium. 1st report on modelling, evaluation and testing for xtreemos security assurance. http://www.xtreemos.eu/publications/project-deliverables/d3-5-10.pdf, January 2009.

[3] XtreemOS Consortium. Evaluation report. http://www.xtreemos.eu/publications/project-deliverables/d4-2-6.pdf, December 2009.

[4] XtreemOS Consortium. Fourth specification, design and architecture of the security and vo management services. http://www.xtreemos.eu/publications/project-deliverables/d3-5-13.pdf, December 2009.

[5] XtreemOS Consortium. Security services prototype month 36. http://www.xtreemos.eu/publications/project-deliverables/d3-5-12.pdf, June 2009.

[6] XtreemOS Consortium. Distributed xtreemos infrastructure (dixi). http://www.xtreemos.eu/publications/project-deliverables/D3.2.17.pdf, March 2010.

[7] 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.

[8] David L Groep, Michael Helm, Jens Jensen, Milan Sova, Scott Rea, Reimer Karlsen-Masur, Ursula Epting, and Mike Jones. Grid certificate profile. http://www.ogf.org/documents/GFD.125.pdf, March 2008.

[9] 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.

[10] http://exist.sourceforge.net/. exist-db, an open source database management system.

[11] P. Menage. *Linux documentation: Control Groups*, 2008. Available at `http://www.mjmwired.net/kernel/Documentation/cgroups.txt`.

[12] Commitee Specification. Oasis standard, extensible access control markup language, August 2003.

[13] 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.