Project no. IST-033576

XtreemOS

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

Second Specification of Security Services
D3.5.4

Due date of deliverable: 30/11/2007
Actual submission date: 14/11/2007

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

Type: Deliverable
WP number: 3.5
Task number: T3.5.2/T3.5.3

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

Version 1.0 / Last edited by Erica Yang / 14/12/07

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

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.0	07/09/07	Erica Yang	STFC	first outline
0.1	25/09/07	Erica Yang	STFC	incorporated updates from telco
0.2	01/10/07	Daniel Vladusic	XLAB	first version of Accounting Service
0.3	08/10/07	Erica Yang	STFC	moved federation to future work, move APIs to D3.5.5
0.4	10/10/07	Erica Yang	STFC	added glossary and 1st draft of security
0.5	02/11/07	Erica Yang	STFC	first version of security services and security architecture for AEM
0.6	10/25/07	Gregor Pipan	XLAB	added policy management AEM use case
0.7	19/10/07	Adolf Hohl	SAP	added initial version of evaluation
0.8	10/25/07	Philip Robinson	SAP	added design principles and integrated the Trust Model with the System Model
0.9	03/11/07	Haiyan Yu	ICT	added node level policy service
0.9.1	03/11/07	Erica Yang	STFC	total reorganization and final checking for internal review
0.9.2	12/11/07	Erica Yang	STFC	revisions based on 1st internal review
0.9.3	16/11/07	Erica Yang	STFC	restructuring for final submission, incorporating Philip's and Adolf's comments and suggestions
0.9.4	20/11/07	Erica Yang	STFC	added functional overview of XtreemFS and AEM,
0.9.5	26/11/07	Erica Yang	STFC	nearly final structure and revised security architecture for AEM and XtreemFS
0.9.6	30/11/07	Erica Yang	STFC	revised concepts and VO management chapter
0.9.7	05/12/07	Erica Yang	STFC	conclusion, executive summary
0.9.8	05/12/07	Philip Robinson	SAP	revised architectural discussion
0.9.9	07/12/07	Philip Robinson	SAP	revised the entire Evaluation
0.9.11	07/12/07	Erica Yang	STFC	nearly final security architecture for XtreemFS
0.9.12	07/12/07	Erica Yang	STFC	removal of accounting
0.9.13	07/12/07	Julian Gallop	STFC	Restructured the evaluation/techincal discussion Chapter
0.9.14	10/12/07	Yvon Jégou	INRIA	enhancements and clarifications on concepts and security architecture
0.9.15	14/12/07	Erica Yang	STFC	final revision on executive summary, conclusion and revisions based on Julita's comments
1.0	14/12/07	Erica Yang	STFC	final version with small corrections and proofreading

Reviewers:
Luis Pablo Prieto (TID) and Julita Corbalan (BSC)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T3.5.2	Specification of XtreemOS Security Services	STFC*, SAP, XLAB, ICT, ULM, INRIA
T3.5.3	Autonomic Security Policy Management and Enforcement	STFC*, SAP, XLAB, ICT, ULM, INRIA

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

Executive Summary

This is a second draft specification of security services for XtreemOS. The main objective of this document is to describe a security architecture that makes use of the security services to secure application execution and file management in XtreemOS.

To set the context, we first present the background to this deliverable. This consists of two parts: (1) the basic functional overview of AEM and XtreemFS; and (2) the security challenges arising from application execution and file management in XtreemOS and those arising from deploying, managing and running legacy applications in XtreemOS. Based on this understanding, we describe the key concepts involved in a XtreemOS VO environment. The notion of global entities is introduced to describe the entities persisting in a global namespace. Three kinds of global entities are described, they are: VOs, users and resources(e.g. machines/nodes) in a VO. The conventional Linux operating system is *not* aware of any of them. Therefore, we describe in details the management issues (e.g. identity, attribute, membership, and policy) of these entities in XtreemOS, and, more specifically, how a standard X.509 certificate is extended as a vehicle to carry the global identifiers and attributes of global entities to allow the operating system to interpret them in the operating system's local namespace. Once the link between the global namespace and the local (i.e. operating system) namespace is established, a security architecture is presented to demonstrate how the security services, being developed and refined in this specification, can be used to provide three basic security mechanisms (i.e. authentication, authorization, and delegation) to achieve secure application execution and file management in XtreemOS, a Grid operating system with native support for VOs.

The last part of this document presents a range of insights which we have gained along the way of producing this deliverable. Such insights are grouped into two categories of discussions. First of all, we present a discussion on the security architecture principles that have been derived by considering requirements from both business and scientific applications and the management of the infrastructure supporting them. It is expected that this discussion can evolve into a set of useful design guidelines for the future edition of the security architecture and could give an early indication of the pros and cons of the the ongoing security design. Next, the discussion on the *new* requirements and challenging issues arising from enabling (i.e. installing, running, and managing) *complex legacy applications* in XtreemOS is presented. It captures our still developing understandings and knowledge of the role played by and the functionalities provided by various WPs (i.e. WP 2.1, 3.3, and 3.5) to realise such advanced applications in XtreemOS. The aim of this discussion is to provide useful inputs to the next stage of the work that our WP will undertake.

Contents

Glossary

AEM	Application Execution Management
CDA	Credential Distribution Authority
CA	Certification Authority
GGID	Global Group IDentifier
GUID	Global User IDentifier
GVID	Global VO IDentifier
NLP	Node Level Policy
PDP	Policy Decision Point
PKI	Public Key Infrastructure
TCB	Trust Computing Base
VOM	Virtual Organization Management
VOPS	Virtual Organization Policy Service
XtreemFS	XtreemOS File System
XOSD	XtreemOS Daemon

List of Figures

List of Tables

1 Introduction

This deliverable is a second draft edition of XtreemOS security services. It is a refinement and revision of its previous edition with an emphasis on presenting an overall security architecture for XtreemOS. In many senses, the work undertaking by the WP3.5 partners is very demanding as it requires not only technical knowledge and expertise of security and VO management in Grids, but also a good understanding of the entire XtreemOS system, not to mention that the former is such a vast research area and XtreemOS, itself, is a large project still in active development and evolvement.

1.1 Brief Recap of D3.5.3

We have shown our awareness and understanding of the technical aspects of XtreemOS and their security requirements in D3.5.3 [5] by describing the technical background to WP3.5, including:

- Linux security,
- Isolation and sharing issues in VOs, and
- Function description and security requirements of various aspects of the XtreemOS system, including:
  - Kerrighed,
  - Application checkpointing,
  - Federated resource management,
  - Highly available and scalable services,
  - Application Execution Management (AEM), and
  - Data management (XtreemFS)

Also in D3.5.3, an initial list of the security services was presented along with a selective range of XtreemOS use cases. Supported by the security services, an initial list of security mechanisms (i.e. mutual authentication, authorisation, secure communication, and isolation) were examined. However, what has been lacking is a big picture - a big picture of the XtreemOS security architecture that brings all the security services together to secure the workflow of the entire XtreemOS system.

In XtreemOS, jobs and files are the first-class citizens. Two technical work packages, Application Execution Management (AEM) and Data Management (XtreemFS) are designated to deal with these entities. Thus, the main aim of the current deliverable is to deliver an *in-depth and focussed* presentation of a security architecture for application execution and file management.

1.2 Document Organization

The rest of the document is organized as follows.

Chapter 2 describes the technical background to this deliverable. It starts with a functional overview of AEM and XtreemFS. This is followed by the presentation of a list of security requirements from their perspective and those from (complex legacy) applications' perspective.

Chapter 3 introduces a set of concepts (e.g. entities, identity, attributes, membership, policies) in a XtreemOS VO environment to lay the foundation for presenting the security architecture.

Chapter 4 presents the XtreemOS security architecture for application execution and file management. This chapter consists of a detailed description of the security services and how they are used to secure AEM and XtreemFS.

Chapter 5 first presents a set of security architecture design principles that have developed to give us early understanding of the limitation of the current security design. It then describes the *new* security requirements and issues arising from deploying, running and managing legacy applications in XtreemOS.

Chapter 6 concludes this deliverable and outlines the future directions for the next edition of this specification.

Priorities of this deliverable A range of tasks (i.e. T3.5.2[1], T3.5.3[2], and T3.5.4[3]) are being undertaken by WP3.5 partners between month 13 and 30. Some areas of work, such as T3.5.3 and T3.5.4, are being studied on a longer timescale and therefore will have a less complete treatment in this document. Thus, in the following, we present a detailed list of work with their priorities in the current deliverable to capture the ongoing progress made in our WP:

- *Top priority:* specification of security services, security architecture, policy management, VO creation
- *Medium priority:* federation of security domains
- *Low priority:* virtualization and levels of isolation. *XtreemOS should provide the possibilities to incorporate various virtualization technologies, such as VMware, Xen, KVM (Kernel-based Virtual Machine), rather than develop its own virtualization support.* We can analyze and evaluate, but not develop the support for strong isolation technologies.

Material Relocation and Sister Deliverables We have presented a lightweight mutual authentication protocol for mobile devices in the previous specification -

[1] Specification of XtreemOS security services

[2] Autonomic security policy management and enforcement

[3] Federation of distinct and autonomous security/administrative domains

D3.5.3. This protocol has been refined and strengthened since. Specifically, we have produced the formal proof for this protocol, which has been verified with well known security protocol checker to ensure its security properties (integrity, confidentiality, availability) against a range of security attacks, such as man-in-the-middle attacks and masquerading attacks. The revised protocol, with its proof and verification, are now part of the formal analysis document (D3.5.6).

Apart from D3.5.6, there is another deliverable D3.5.5 from WP3.5 which is submitted as a companion to the current deliverable. D3.5.5 presents a up-to-date view on the current prototype implementation.

2 Background

The purpose of this deliverable is to specify security services for XtreemOS. In this chapter, we first present the outcome of our ongoing study into the functionalities of AEM and XtreemFS. As XtreemOS is still being actively designed and implemented, we have tried all our best to reflect on what is being made available through existing deliverables and our active and in fact, still ongoing, interactions with other WPs.

We begin with showing how they work independently. Then, we describe how they interact with each other to create a complete picture of the basic functionality of XtreemOS: from job submission to accessing files. The security mechanisms are purposefully left out from the function description. To avoid dealing with the internal details of AEM and XtreemFS, we have concentrated on: (1) identify key components in the subsystems; and (2) understanding the interactions between the node level services and the global services.

The second part of this chapter looks at the security challenges introduced by the design of AEM and XtreemFS subsystems. We also present a list of security challenges arising form studying legacy applications in the context of XtreemOS.

Next, we will start by explaining briefly the functional overview of AEM and XtreemFS, and, the interactions between them.

2.1 Functional Overview of AEM and XtreemFS

This section is divided into three parts: how AEM works, how XtreemFS works, and how AEM can interact with XtreemFS. We should emphasis that this deliverable is not the ultimate source to grab the details of the internal working of these systems because:

- Their internal details have been abstracted away to serve the purpose of our deliverable, i.e. presenting an overall security architecture for XtreemOS without getting into the detailed interactions inside these systems; also
- Some interesting features (e.g. replication management in XtreemFS) have been omitted to keep the focus of the current deliverable.

Interested readers should refer to the respective deliverables (e.g. [3], [1], [7]) for an in-depth description of them.

The diagrams in this section follows the same pattern. Services have been grouped into two categories: node and global services.

- **Node services** run on each XtreemOS node, be it client or resource node.
- **Global services** run on a global scale (i.e. a VO-wide scale) serving all the nodes in a VO.

How AEM Works?

According to [3], AEM components are divided into two parts: client side (interface) and server side (service). Depending on their location, AEM services are further divided into two sorts: node (level) services and external (global) services. In the context of AEM[4], global means VO-wide.

Figure 1 illustrates the interactions between the AEM components. In this diagram, the client side is represented by the XtreemOS console, which is (currently) a command line interface to allow users to type in AEM commands, such as *Xsub* for job submission, and *Xps* for querying current jobs.

XOSD is a server-side daemon service running on *each* XtreemOS node for managing jobs (e.g. submission, execution, monitoring), and resources (e.g. allocation and negotiation). In Figure 1, it is shown running on a client node as *XOSDc* and a resource node as *XOSDr*. XtreemOS console and XOSD are node (level) services.

AEM has two distributed services (classified as external services in the figure): JobDirectory and ResMat (ResourceMatching). JobDirectory provides job query services to users to allow them retrieve the JobManger's address for their jobs. ResMat is responsible for finding resources (machines) that can satisfy the criteria in the job specification and have spare resources (e.g. CPU and memory) to provide at the time of job submission.

In the following, the AEM job submission process is selected to illustrate the interactions between AEM components as it is the most typical AEM functionality. Figure 1 shows that there are 12 steps involved in a job submission process.

1. A user types the *Xsub* command to the XtreemOS console. A simplified example is shown below:

   ```
   Xsub -vo esvo -group testing -role programmer -f job.jsdl
   ```

   It means that the user who is a programmer in the testing group wants to submit a job, with the job specification job.jsdl, to a VO called esvo.
2. XOSDc generates a jobID and creates a job with the jobID
3. XOSDc registers the job to the JobDirectory with jobID, @JobMng address
4. registration: success or failure
5. once the job has been registered, XOSDc starts scheduling the job by contacting ResMat with a list of requirements
6. ResMatching returns a list of resource nodes to XOSDc.
7. XOSDc performs a resource selection algorithm to decide which (set of) nodes to be used.

[4] In XtreemFS, global has to be truly global, i.e. within a VO and across VOs.

8. XOSDc negotiates resources with XOSDr on the selected nodes until all the required resources (e.g CPU, memory, queue ...) are reserved.
9. The reservation confirmation is then sent back from XOSDr to XOSDc.
10. Once all resources are reserved, XOSDc submits the job to the corresponding (set of) XOSDr.
11. XOSDr forks a user process *UP* for the job. The job will then start its execution on the node. This completes the job submission process.
12. The confirmation of job submission is sent back to XOSDc.

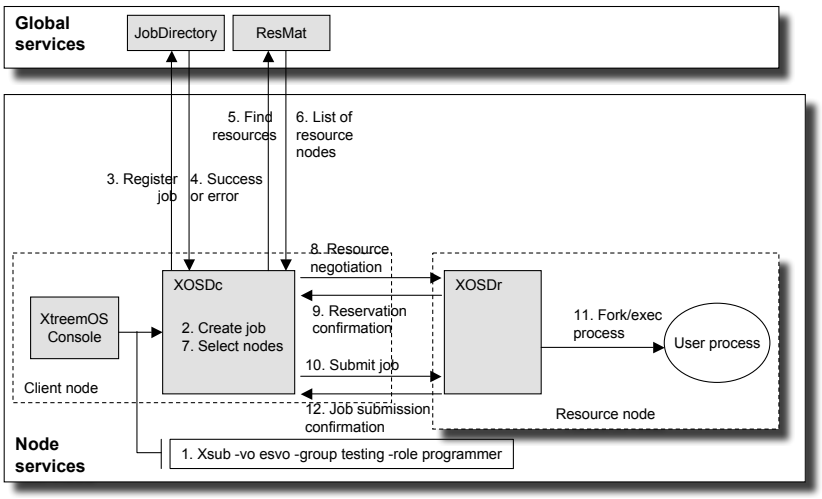

Figure 1: An Overview of Interactions within AEM

How XtreemFS Works?

XtreemFS is an object-based distributed file system. In addition to distributed file accessing, it also offers advanced file management features, such as facilities to enable inter-process communication for applications (Object Sharing Service), and fault-tolerant file replication service (Replica Management Service). Like the previous section, we will concentrate on the most typical use of XtreemFS - accessing XtreemFS files (without the additional features).

Three components are typically involved in accessing (i.e. create, delete, read, write) a XtreemFS file. They are:

- Metadata and Replica Catalog (MRC)

- Object Storage Device (OSD)
- XtreemFS client (XFS client)

To be self-contained, we have included the description of MRC, OSD and XFS client from the WP3.4's deliverable - D3.4.1 [1].

The **MRC** *is responsible for maintaining all file system metadata, extended (user defined) metadata as well as information on replica locations. It also hosts access control policies and makes authorisation decisions.*

The task of the **OSD** *is to provide functionality for data access in the file system. It offers an object-based storage interface to hide the complexity associated with underlying block-based storage mechanisms. Capabilities of the component include read and write access, concurrency control and communication with remote storage hosts.*

(XtreemFS) Clients *are hosts running components of the access layer, i.e. the file system adapter or the XtreemFS library. Applications and user processes use the access layer to communicate with XtreemFS components. This can be done transparently to the application through the traditional Linux file system interface. XtreemOS aware applications can take advantage of the native XtreemFS interface through a library provided by the access layer.*

Figure 2 illustrates the interactions among MRC, OSD, and XFS client (aka. XFSc). The following interactions between XtreemFS components are originated from our discussion with WP3.4. Specifically, it was first presented in an XtreemOS internal workshop between WP3.5 (security) and XtreemFS [9]. In the figure, we have abstracted away the *internal details* of XtreemFS, such as how XtreemFS makes use of the Virtual File System (VFS) to implement a uniform file access interface. Instead, we continue to use XFSc to represent all these internal complexity of XtreemFS. Therefore, from the perspective of WP3.5, the following steps are involved in a typical file operation with XtreemFS:

1. A file (operation) request (e.g. create, open, delete, read, write a file), initiated by a user (application) process, comes into the XFSc via the Linux kernel[5].
   - 1.1 The XFSc requests local credentials from the FUSE daemon by passing in the PID of the calling process
   - 1.2 The XFSc receives UID/GIDs back
2. The XFSc asks the MRC whether the operation is permitted.
3. If the operation is allowed, the MRC returns with a set of capabilities. Otherwise, the operation is deemed as denied.

[5]This kernel must be compiled with the Filesystems in Userspace (FUSE) kernel module and support the Virtual File System. See the deliverable D3.4.1 [1] and D3.4.2 [7] for more details

4. With the capabilities, the XFSc contacts the appropriate OSD(s). (Only one OSD is contacted in the diagram to illustrate the scenario. But in reality, a file may be stored in multiple OSDs.)
5. The OSD(s) transfers the file to the XFSc. If this process fails, errors will be reported.
6. XFSc presents the file back to the user process.

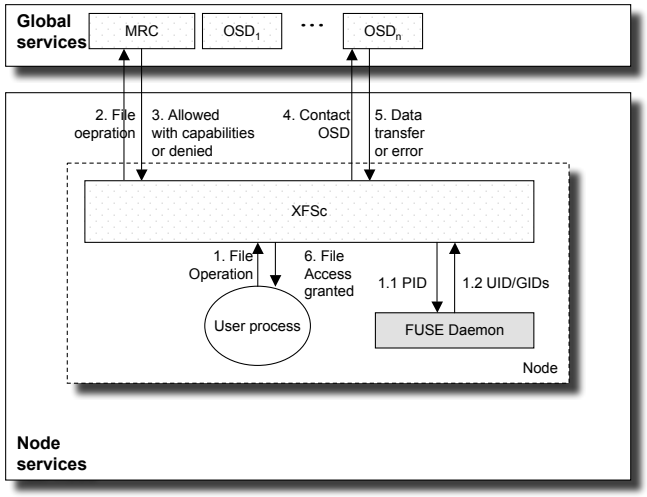

Figure 2: An Overview of Interactions within XtreemFS

How AEM Interacts with XtreemFS?

What has been shown in the previous section is the interactions among XtreemFS components for processing file operations. As with any other distributed file systems (e.g. NFS), XtreemFS can also be used to support application execution. This section presents how AEM interacts with XtreemFS during a job submission process in an XtreemOS envionment.

Figure 3 is a figure composed from Figures 1 and 2. To our understanding, the link between AEM and XtreemFS is via user processes[6]. Such processes can be interactive processes that connect to a terminal or batch processes run without a terminal. Also, how such a process is running (foreground or background) is irrelevant to XtreemFS.

User processes are typically associated a user's (real) UID, which is given by the local operating system where the user is logged into. However, when a user logs in to a client node (as shown on the left hand side of Figure 3), the process's

[6]Other interactions between these two are still being studied in the project.

UID (shown on the right hand side of the figure) is irrelevant to the login process on the left hand side. There comes a question how a resource node associates the correct credential to a user who logins from a client node, when there is no existing account created for the user on the resource node?

Before exploring the solutions (in Chapter 4), the next section shall summarise the security challenges that arise in facilitating the internal interactions within AEM and XtreemFS respectively, and the interactions between them.

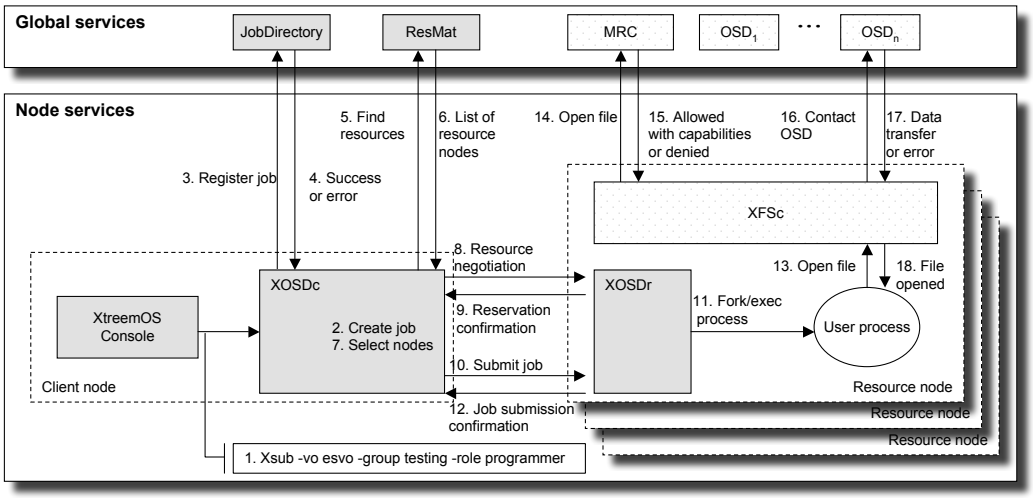

Figure 3: Interactions between AEM and XtreemFS in XtreemOS

2.2 Security Challenges

2.2.1 The AEM and XtreemFS Perspective

In the last specification, we have elaborated a number of security requirements for AEM and XtreemFS. This section is intended as a replacement for these requirements based on our current (already evolved) understanding of AEM and XtreemFS, which has advanced since the last deliverable.

AEM and XtreemFS are supported by VO management (aka. VOM) services. VOM is a logical representation of a collection of VO infrastructure services, which include identity, attribute, membership and policy management services for VO users and resources. Here, we focus on three key categories of security requirements: authentication, authorisation, and accounting.

Authentication Users need to be authenticated to access the *global services*. In the context of AEM, it means that only a user with valid VO credentials can access

JobDirectory and ResMatching services. In XtreemFS, it means that such users need to be authenticated while accessing the MRC and OSDs.

The authentication to the *local* node services, including the XtreemOS console and XOSD of AEM, and the XtreemFS client, relies on the local mechanisms available on the node from which a user's request (e.g. job submission or file accessing) is initiated[7]. Typically, this can be either the Linux account login procedure of a Linux box. Or, if the node is part of an organizational network, it can be a network-based authentication system, such as Kerberos. Regardless how a user gets access to a XtreemOS node, the node authentication procedure should ensure that a user has a valid UID/GID(s) on the node.

Node services, specifically XOSD, can be accessed remotely, for example, to support job execution. The authentication with remote node services needs to be supported by mutual authentication, where entities on both ends should be authenticated. However, in a large scale distributed environment (consisting thousands of nodes) that XtreemOS targets, it is unrealistic to assume that users have existing accounts on remote nodes. Therefore, the challenge is to manage such a trust relationship without compromising the scalability (the number of users and resources in the system) of the system.

Meanwhile, as XtreemOS jobs can be interactive and batch, it is essential to ensure that (1) users' jobs do not interfere with each other; (2) users' files are protected with sufficient access control mechanisms; (3) users are attached to the correct sessions of application execution; (4) users are given the same level of privileges in all the sessions; and (5) once a job is complete, users' credentials should be safely revoked and their privileges be removed across the entire VO. However, upon job completion, users should still be able to access to the results (e.g. the files) with appropriate rights and should even possibly grant these rights to other users.

Authorisation Fine-grained access control mechanisms should be in place to ensure AEM and XtreemFS services are available to authorized users *and* they can perform efficiently even in the presence of a large number (e.g. thousands) of users and resources.

Users need to be authorised to gain access to VO resources (nodes and files). At the VO level, this means that VO resources should be coordinately managed by the VOM services and be shared among a large number of users. It should be possible to allow VO owners and managers to specify how they would like to see their resources be used. However, the ultimate control to machine resources (e.g. CPU, memory, file space) should be with the hands of resource owners who rely on local operating system level access control mechanisms (e.g. UID/GIDs and

[7]This has been referred as the Home Authentication Authority in D3.5.3.

the Discretionary Access Control model) for files and processes.

Accounting In the context of AEM, the security services for job resource accounting should ensure that resource usage is recorded with accuracy so that resource consumption is accountable. By accountability, we mean that the accounting information should associate with individual *VO* users. The recorded information should also be made available to authorized VO entities so that they can run follow-on services (e.g. reputation or billing) after a job is finished.

2.2.2 The Application Perspective

This section looks at security challenges considered in the XtreemOS security architecture from an applications perspective, extending those identified specifically for AEM and XtreemFS. We define applications as a collection of interconnected software and data for a specific purpose and set of transactions, distributed over a network of compute nodes. We also assume that the nodes are connected by an insecure network and that each node is a machine running XtreemOS. Secondly, each node is *trusted for local security* by their organizations, users and applications, meaning:

- any node is computationally capable of performing sufficiently strong cryptographic procedures for authentication, privilege assignment, authorization, encryption/decryption of data and maintaining integrity of transaction logs;
- if only local administrators and users of an organization have accessed the node directly, then all users and applications are expected to be trusted and legitimately using resources; and
- the node has a reliable mechanism that ensures that users can only access and processes data to which they have locally specified rights, including ownership and membership in an organization.

The above are not assumed when nodes are accessed over a non-trusted network, moreover across organizational boundaries, but must be enforced by the appropriate security mechanisms. Even if the nodes are running in a single, physical organizational domain, for example a data-center, the cross-organization network is treated as non-trusted, as the executables and data objects hosted on each node (and even the same node) could be from different users and even organizations (such as in the case of a data center). Furthermore, future Grid environments will need to offer additional services that expose some data for monitoring, billing and accounting of the resources allocated to them. The security architecture needs to

provide the assurance that privileged users or administrators can only view the collective resource usage of the set of nodes currently and potentially available to host their application elements and data objects. These problems will be increasingly significant for business and scientific applications in a Grid execution environment. We identify application-related security challenges as dealing with a differing application security requirements, complex user management and securing the local and distributed runtime of applications.

Securing different Applications of different Organizations The XtreemOS security architecture is designed to secure applications with different types of needs, ranging from scientific to business. The goal from a scientific perspective is to aggregate compute power for simulations, while from a business application perspective the goal is to make data and applications more available as the user base (employees, customers or partners) is more distributed and increasing in size. The aggregation of compute power means that simulation data will be split in parts and distributed across several nodes possibly in several different organizations. Similarly, in order for a business organization to maintain better response times for its employees, customers and partners, it may contract third-parties to host different data processing services and applications. These parties are then liable for ensuring the security properties required by the application and data owners. Data should only be read and written by legitimate parties and be available to relevant, external parties.

Within a Grid execution environment, trust and networking relationships will be formed and destroyed dynamically and continuously over time. Relationships will also be created between the members of these physical organizations. The security challenge here for XtreemOS is to provide distributed OS components and mechanisms to ensure that organizational relationships, represented by communicating processes or machines, are correctly represented and maintained. That is, data exchanged during the creation and maintenance of these relationships should only be accessed, modified and read by the participants involved. Moreover, the computational resources used to support and maintain an organization's relationships with different partners should not impact on the maintenance of others.

Each organization will also have different users, who may act on behalf of the organization. XtreemOS needs to provide security components and mechanisms that can assert and validate the affiliations of different users, such that policies based on organizational affiliation can be enforced. For example, employees of an institute can only access information stored on a remote machine (e.g. OSD) if they have the requisite credentials to prove their employee status. In addition, users may also be registered as members of VOs, such that there also needs to be mechanisms to prove these cross-organization affiliations.

Each organization will also use and provide individual machines, storage facilities and, in some cases, personnel in order to support collaboration and resource sharing. In order to avoid rogue nodes being introduced into a shared utility or network of compute resources, XtreemOS will have to provide security mechanisms that can verify that a node has been offered by an organization for sharing as well as to assure users that they are accessing valid target resources. In a network of possibly 10s of 1000s of nodes, this validation of organizational authenticity will still have to occur quickly but assuredly. Even in cases where complex, virtual overlays or a distributed file system (XtreemFS) are used, organizations will still want to avoid their sensitive data being stored on physical machines of competitors. Even if a malicious competitor cannot read the data, but is aware that their resource is being used for a transaction by a competitor, they can easily interfere with or corrupt the transaction and/or data.

In addition to machines, the software owned and licensed by organizations will also be considered to be valued resources in the network of services enabled by XtreemOS. Organizations will want to have mechanisms for proving the ownership or origin of application software or jobs running on distributed resources. From an organizational perspective, this is critical for performing effective accounting. XtreemOS must therefore provide adequate security mechanisms that enable this form of binding applications to their respective owning organizations, such that the accounting procedures can be easily implemented and executed.

Secure user management for applications User management is a second challenge for the XtreemOS security architecture. For AEM, it is important that remote users can be authenticated as well as tracked with respect to their resource usage. For XtreemFS, there are several challenges for representing and authenticating globally distributed users sharing a distributed filesystem resource. Unlike AEM, the filesystem is considered to be part of the so-called F-Layer, for the most part, such that it needs to consider the constraints of this layer. These include the constraint on valid user names and user-ids (UIDs) in Unix-based operating systems. Users also need to be globally and uniquely distinguishable, either at the application or OS layer. The security mechanisms of XtreemOS will have the responsibility of supporting this uniqueness at both layers, as applications will depend on this functionality being in place.

Users of resources will only be able to access them after being able to prove membership or affiliation with trusted organizations contributing to a set of distributed compute resources. Users will login to different client machines that will inevitably be responsible for maintaining some of their local data and credentials. The management of a large set of users is hence a challenge. A reason for this is scalability and availability of an identity management infrastructure and also the

management of user identities and rights in systems among different user management concepts. E.g. Unix uses the notion of users and groups to describe right and thus access control to files. This was appropriate for the context it was made for. However it cannot cope easily with today's needs. More recent derivatives provide the concept of roles and flat access control lists as the Windows operating system does.

However most legacy applications rely on the user/group notion to achieve security by the concept of delegation of duties. Attackers have multiple barriers to break in order to reach their goal. This also helps to prevent the propagation of errors. Even if application security does not rely on the user concept solely it is used as an additional barrier.

Regarding business applications, this concept is used to execute different parts of the application, e.g. the database in the context of a different user. The following figure shows eight default application users of a standard SAP business application.

```
sdb:x:1000:1001:Database Software Owner:/home/sdb:/bin/bash
lcaadm:x:1001:1001:Owner of Database..:/home/lcaadm:/bin/csh
pv4adm:x:1002:1002:System Administrator:/home/pv4adm:/bin/csh
db2pv4:x:1003:1003:Database Administrator:/db2/db2pv4:/bin/csh
apppv4:x:1004:1005:Database Connect User:/home/sappv4:/bin/csh
apppv4db:x:1005:1005:Java Database..:/home/sappv4db:/bin/csh
appadm:x:1006:1002:System Administrator:/home/sapadm:/bin/csh
trxadm:x:1007:1002:System Administrator:/home/trxadm:/bin/bash
```

The set of users are created during the interactive installation procedure. It has to be pointed out here, that most Unixes do not support fine grained authorization in order to enable a non-fully-privileged user but root to perform the management setup of additional users. During that phase, consider that some 200.000 files need to be put into place. A set of characteristic system parameters have to be set and very often databases are connected directly to raw devices. Most of these actions also require the interaction of the privileged root user. This also has implications for the success of the overall application runtime.

Securing the Application runtime Emerging Grid applications for business and science are not simple software bundles, but they are typically composed of various components and stand-alone utilities for user interface rendering (i.e. presentation/web servers), execution of workflows and solution logic (i.e. application servers) and maintaining queries and persistence of the critical data that continues to grow with every transaction.

Each server would then have a different configuration based on the types of data, processes and connections they need to handle. Configurations of the operating system may be complex, such that doing them incorrectly may lead to

operational conflicts, inconsistencies and undesirable application downtime. Another consideration is if there is a need to move applications from one node to another as a result of failure, predicted load increases, test-to-live transition, new hardware or changing partnerships. Being able to migrate operating system instances across distinct physical nodes is a useful tool for administrators of data centers, clusters and even distributed Grid nodes: It allows a clean separation between hardware and software, and facilitates fault management, load balancing, and low-level system maintenance

Application management is a process that starts from installation of an application and proceeds to its execution, monitoring, adaptation and eventual termination. Adaptation may include resource configuration setting, checkpointing and migration. The security challenges for each of these steps are discussed below:

Application installers require privileged root access to target hosts in order to customize and tune the machine for interoperability and efficient support of the applications and data management solutions being installed. They need to query, set, disable and enable certain variables in the registry, according to the setting stored for the operating system in the system library (syslib). These include access to the following:

- accounts: creation of users and groups, and manipulating access control lists
- local filesystem: creating installation files and directories, copying and deleting them as well as working with mount points and shares.
- network: retrieving the IP address of the network card, registering services (e.g.in the /etc/services file) and altering the IP tables in the /etc/hosts file.
- process: manipulating context of current process of external programs from which information is required
- system: retrieval of hardware information including RAM, storage, num processors, processor speed, storage type, BIOS
- timer: use of time functions for logging, timing and scheduling installation events

Furthermore, in order to perform these actions, the installer needs to assume a standard interface to the OS for performing these actions. In the case of a UNIX-like OS, this interface would need to be POSIX compliant. POSIX-compliance also places some constraints on the way in which users, groups and access controls are organized. Granting privileged root access to installers of enterprise solutions on various machines must be done within introducing security holes in the

systems. However, the intention of using a Grid-like infrastructure is to have this done dynamically, for many different customers, relatively fast and over the wire, such that configuring authorizations and performing installation with too much overhead and manual administrator intervention is undesirable. Furthermore, installation is not a one-off batch job but includes ongoing monitoring and updates of the installation. Solutions for user management need to be scalable, usable and persistent without exposing the target host and its local network to significant risks. Even if the installer were to be malicious, while having privileged access to the host, the impact and propagation of its attacks need to be contained.

3 Entities for Secure VO Management

This chapter describes a set of entities involved in an XtreemOS system environment and how the identity, attributes, and policies of these entities are managed by the system. In the next chapter, we shall describe how they are used by the security services to provide authenticated and secure interactions between entities and a cascading and scalable access control framework to create a secure application execution environment for AEM and enable a secure file accessing infrastructure for XtreemFS.

3.1 Global vs. OS Entities

In an XtreemOS Grid environment, there are two types of entities: global entities and OS entities. Global entities include VOs, users and (resource) nodes in a VO. They persist in a global namespace and are identified by a public key certificate.

In the operating system, there are two types of OS level entities: OS users and OS resources (files and processes). They exist in a OS (local) namespace. In the OS, users are identified by a U(ser)ID, files are identified by an inode number[8]; and processes are identified by a P(rocess)ID.

Conventional operating systems do not provide native support for the global entities. Thus, the key challenge is to enable the operating system to recognize the global entities by providing a mapping between these two, *in particular*, the mapping between global users and OS users.

3.2 Actors in a VO

In an XtreemOS VO, there are a number of actors involved in a VO's lifecycle:

- **VO creator**: A VO creator, a person or a service, is the one who creates a VO. It has the following responsibilities:
  - specifying the followings, called a VO specification, to set up a VO
    - VO attributes (e.g. a list of group names, role names, subgroup names, capabilities) - compulsory
    - (the private key and public key certificate of) the VO manager - compulsory
    - a set of VO policies - optional
    - a set of VO members -optional
  - managing the lifecycle of the VO

[8] inode is a data structure which contains information about a file in a Linux/Unix operating system.

- **VO manager**: A VO manager is an entity (either a person or an organization) liable for the authenticity of identity and attributes (stored in a public key certificate) of global entities. A VO manager is identified by its public key certificate and its private key is used to sign certificates for global entities. A VO manager can issue user certificates for multiple VOs.
- **VO administrator (a.k.a. VO admin)**: A VO admin, operating on behalf of the VO manager, is an abstraction of a set of software programs/services responsible for administrative tasks in a VO, including
  - running XtreemOS security and VO related services, as described in Section 4.1, which entails
    - installing the VO manager's private and public keys to the correct location
    - generating the *structure* of VO databases
    - signing user certificates using the VO manager's private key
  - managing a VO, which entails
    - generating and managing attributes for users and resources of the VO, upon registration
    - generating a certificate for users
    - specifying policies for the VO
    - removing users and resources from the VO
- **VO member:** VO members are users (consuming resources) and resources (providing resources) in the VO. VO members can have identifiers and attributes allocated by the VO admin.
- **Resource Administrator (a.k.a. resource admin)**: A resource admin has the final control over the resource provision on its node and and its trust relationship with VOs. Specifically, a resource admin is responsible for
  - installing the node's private and public keys to the correct location
  - configuring the node appropriately (such as admitting the VO manager's public key certificate into its trust anchor), once being admitted into a VO
  - running node services
  - specifying policies for the resource
  - ensuring the authenticity of resource usage information it supplies
  - registering the resource to a VO

These actors are logical groupings by their responsibilities. In practice, one, be it a person or a software service, can take up the responsibilities of one or more actors. For example, a person can simultaneously become a VO creator, a VO member, a VO administrator, and a VO manager. In the very extreme case, a

person can have the responsibilities of all these roles. The person (or a service) who creates a VO can become a VO member, a VO administrator, a VO manager, and a resource administrator, given that he also provides resources to the VO.

3.3 Identity, Attributes, and Membership

This section specifically describes the identity, attributes, VO membership associations, and policies that global entities can have.

3.3.1 Identity of Global Entities

In XtreemOS, each global entity is identified by a X.509 v3 public key certificate [10] which certifies the binding between its identity and its public key. The structure of a X.509 v3 public key certificate is shown as follows:

- version (v3)
- serial number
- signature algorithm
  - identifier
  - parameters
- issuer's DN
- validity period
  - not before date/time
  - not after date/time
- subject's DN
- subject's public key
  - algorithm identifier
  - parameters
  - public key value
- issuer unique identifier (optional)
- subject unique identifier (optional)
- extensions
- certificate signature (of the entire certificate, including the extensions)

The subject's Distinguish Name (DN) of a global entity's public key certificate can uniquely identify the subject in a PKI world, thus becoming the identity of the subject. Once the DN is assigned to a global entity, the association should remain the same throughout its lifetime.

This unique association between a global entity and the DN field in its certificate applies to all global entities. For instance, when the identity of a VO is generated, it should be maintained throughout the VO lifecylce. Similarly, the DN of a user's or a node's certificate should remain the same regardless how many VOs they register with and how many different attributes they may have in each VO.

3.3.2 Attributes of Global Entities

All global entities can have attributes associated with them. An *attribute* is a property that an entity has. In XtreemOS, attributes are stored as extensions in a X.509 certificate, called XtreemOS extensions.

Each global entity can have a list of attributes associated with them. Except resources, all global entities have a global identifier - a special type of attribute, associated with them.

Although the DN field in a certificate is a useful source of information about an entity, it is too long to be used as a unique identifier in the operating system. In a X.509 certificate, the DN field consists of the following information:

- Country
- Locality (or city)
- State (or province)
- Organization
- Organization Unit
- Common Name (including email)

In the traditional Linux, the POSIX compliant identifier (local Linux names) is only 8-byte long. In the recent versions of Linux, the length of Linux identifiers (names) has been extended to 32-byte.

Thus, instead of using the full DN field, the XtreemOS mapping process (see [2] for details) uses a 32-byte identifier, called *global identifier* to identify users and map them into operating system level users. Table 2 summarizes the attributes for all global entities. Users can have the following *attributes*:

- Global Vo IDentifier (GVID): a user's VO association
- Global User IDentifier (GUID): a user's Global User IDentifier
- Global Group IDentifier (GGID): a user's global group association
- Group: a user's group association within a VO
- Subgroup: a user's subgroup association within a group
- Role: a user's role in a group
- Capability: a list of capabilities that a user has over certain objects

The exact list of attributes that a user is allowed to have depends on the VO model. It is possible to introduce additional attributes or to just use a selection of them from the above list.

A user can simultaneously hold multiple attributes. For example, when a user registers with multiple VOs, he will be associated with multiple GVIDs. Similarly, a user can register with multiple global groups, and associate with multiple GGIDs. Within a VO, a user can concurrently join different groups and process multiple roles.

There is no global identifier defined for nodes as there is no need to map nodes' identifier to a operating system (local) namespace.

3.3.3 Allocation of Identifiers and Attributes

In XtreemOS, there are three scopes (or contexts) where the identifiers and attributes of global entities can be used. They are:

- XtreemFS scope: *only* meaningful within the scope of XtreemFS
- node scope: *only* meaningful within the resource's operating system
- VO scope: *only* meaningful within the VO

Note that these scopes are only valid within the defined context. The identifiers and attributes become meaningless when the context is changed. For example, even if two VOs, VO_1 and VO_2, both have the concept of groups, the groups in VO_1 are invalid in VO_2, and vice versa. Similarly, if a concept is only valid within the XtreemFS scope, it will not be meaningful outside XtreemFS.

To understand how the identifiers and attributes are allocated in a VO, it is essential to understand who manages them and the scope within which they are used. This is summarized in Table 1.

The GVID for a VO is generated by the VO creator upon the creation of the VO. A GUID is a user's global identifier, allocated by the VO admin, when a user is registered with a VO. Within one VO, it is possible that a user has multiple GUIDs. For example, when a user is associated with multiple groups, subgroups, and roles within one VO, each tuple

```
<group, subgroup, role>
```

uniquely corresponds to a GUID. With different VOs, a user is always associated with different GUIDs. A GGID can be created by a user who has sufficient privileges to perform such operations.

A user and a node can be associated with more than one VO. When more than one GVIDs are associated with a user or a node, the first of such is the primary GVID and the subsequent ones are the secondary GVIDs. Similarly, a user can

Attribute	Created/managed by(when)	Used in scope
GVID	VO creator (when a VO is created)	XtreemFS+node
GUID	VO admin (when a user registers)	XtreemFS+node
GGID	VO user? (when a privileged user requests)	XtreemFS
Group(s) in a VO*	VO creator and admin	node+VO
Subgroup(s) in a group within a VO*	VO creator and admin	node+VO
Role in a group of a VO*	VO creator and admin	node+VO
Capability in a VO*	VO creator and admin	node+VO

Table 1: Who Creates the Attributes and the Usage Scope of the Attributes. (*: the availability of these attributes depends on the VO model. ?: decision pending)

Global Entities	Attributes
VO	GVID
User	GUID, GVID(s), GGID(s), Role, Group, Subgroup, Capability
(Resource) Node	GVID(s)

Table 2: Identifier and Attributes for Global Entities

associate with multiple GGIDs. The first GGID is the primiary GGID, and the subsequent ones will be treated as secondary GGIDs.

Upon registering with a VO, a user specifies the following information:

- attribute values (as shown in Table 2)
- user policies regarding their preferences of job submission (where to or not to submit their jobs) and file location (where to or not to store their files). Such policies will become part of the VO policies (see Section 3.4).

The VO admin then generates GUID(s) and register the user with other attributes, as appropriate.

Global Uniqueness *The uniqueness of GVIDs, GUIDs and GGIDs should be guaranteed globally.* The global uniqueness of GVIDs is guaranteed by the VO creator via probabilistic means. A GUID is a concatenation of a GVID and a VO-wide unique user ID (managed by the VO admin). Hence, the global uniqueness of GUIDs depends on that of GVIDs. GGIDs should also be globally unique, which

can be guaranteed by the VO user via probabilistic means, like generating the GVIDs. However, as GGIDs are independent of VOs, it is a bit more complicated to ensure their global uniqueness.

Open Issues There are other alternatives (e.g. managed by a VO admin or managed by XtreemFS) regarding who can create GGIDs. Like GVIDs, GGIDs need to be globally unique. However, unlike GVIDs, GGIDs do not relate to VOs. As its name suggests, it is global group names, so, it is above all the VOs. Hence, it may not be appropriate to associate them with GVIDs to guarantee its global uniqueness. Thus, we may need to explore the alternatives.

Because GGIDs are only used by XtreemFS, one alternative is to let XtreemFS to create it and feed it back to VOs. This is an open issue that will be resolved in the next edition of this specification.

3.3.4 VO Membership of Users and Resource Nodes

Users and nodes can register with one or more VO(s), thus, having VO membership association(s).

Being a VO member has different implications for users and nodes. The fact that a user is a member in a VO implies that the user has been allocated attributes appropriately in the VO databases. Being a member of a VO, to a node, means that the resource admin of the node, in addition to its normal responsibilities, needs to configure the local policies and the local trust anchor appropriately for this VO.

3.3.5 Certificates for Global Entities

It should be noted that the attributes for all global entities are stored in the VO databases. However, not all of them go into the certificates. The certificates for a VO manager and a node are standard (i.e. without XtreemOS specific extensions) X.509 certificate. The certificates for a VO and a user are X.509 certificates *with XtreemOS specific extensions*. Hence, to avoid ambiguity, we shall now describe the certificate structure for a VO manager, a node, a VO, and a user.

The Structure of a VO manager's Certificate The structure of a public key certificate for a VO manager is described as follows:

- version (v3)
- serial number
- signature algorithm
  - identifier
  - parameters

- the issuer's DN
- validity period
  - not before date/time
  - not after date/time
- the VO manager's DN
- the VO manager's public key
  - algorithm identifier
  - parameters
  - public key value
- issuer unique identifier (optional)
- subject unique identifier (optional)
- certificate signature signed by a known CA

The Structure of a Node Certificate The structure of a public key certificate for a node is described as follows:

- version (v3)
- serial number
- signature algorithm
  - identifier
  - parameters
- the issuer's DN
- validity period
  - not before date/time
  - not after date/time
- the node's DN
- the node's public key
  - algorithm identifier
  - parameters
  - public key value
- issuer unique identifier (optional)
- subject unique identifier (optional)
- certificate signature signed by a known CA

The Structure of a VO Certificate The structure of a public key certificate for a VO is described as follows:

- version (v3)
- serial number
- signature algorithm
  - identifier
  - parameters
- the issuer's DN
- validity period
  - not before date/time
  - not after date/time
- the VO's DN
- the VO's public key
  - algorithm identifier
  - parameters
  - public key value
- issuer unique identifier (optional)
- subject unique identifier (optional)
- extensions (XtreemOS specific extensions)
  - GVID
- certificate signature signed by a VO manager (of the entire certificate, including the extensions)

The Structure of a User' Certificate Given our VO model, a user's certificate has the following features:

- regardless how many GUIDs a user has, the DN field in the user's certificate should remain the same;
- regardless what VO attributes are stored in the XtreemOS extensions in the certificate, the public key of the user and the public key of the VO manager should remain the same; and
- the VO manager's signature varies depending on the information going into the extensions.

The structure of a public key certificate for a user is described as follows:

- version (v3)
- serial number
- signature algorithm
  - identifier
  - parameters
- the VO manager's DN
- validity period
  - not before date/time
  - not after date/time
- the user's DN
- the user's public key
  - algorithm identifier
  - parameters
  - public key value
- issuer unique identifier (optional)
- subject unique identifier (optional)
- extensions (XtreemOS specific extensions)
  - GUID
  - GVID(s)
  - GGID(s)
  - Role
  - Group
  - Subgroup
  - Capability
- certificate signature signed by a VO manager (of the entire certificate, including the extensions)

An Example of VO Attributes in a User Certificate

Meaning of the VO Attributes

- esvo: escience vo
- phvo: physics vo
- chemvo: chemistry vo
- esvou001: escience vo user 001
- esvog001: escience vo group 001

- esvog002: escience vo group 002
- esvog003: escience vo group 003

Samples of VO Attribute Values The following is an list of "mocked" VO attributes that can be embedded in a user's certificate as XtreemOS specific extensions. The string before the semi-column is the human-readable VO attribute name and the string after the semi-column is an sample attribute value.

1. GUID: esvou001
2. GlobalPrimaryVOName: esvo
3. GlobalPrimaryGroupName: esvog001
4. A list of GlobalSecondaryGroupNames : esvog002, esvog003
5. A list of GlobalSecondaryVONames: phvo, chemvo
6. Group: Testing
7. Role: Programmer
8. Subgroup: FacilityTesting
9. Capability: empty

3.4 Policies

Policy is a widely used term in the distributed systems and networking community. Policies are used to describe how an organization or an individual wants to manage their properties (machines, personal data or how their data is being processed).

A policy is, therefore, defined as a statement describing what *actions* a *subject* (or a group of subjects sharing the same characteristics) is allowed to perform on an *object* (or a group of objects sharing the same characteristics) with certain *constrains* (e.g. time, location, limit).

Types of Policies In XtreemOS, there are three types of policies: user policies, VO policies, and node policies, one for each global entity.

Individually, users can define user policies about their *preferences* on how their jobs/files are being treated by VOs. Examples are *"I don't want my jobs to be run on the machines belonging to Organization A"* or *"I don't want my files to be stored on the storage belonging to Organization A"*.

VO polices, defined by a VO admin, have a VO-wide impact on how resources (e.g. nodes and file storage) being utilized by jobs and files, for example, during the process of resource selection and file location/relocation. An example of such is: *"Programmers from organization A can only use resources from organization B between 9am and 5pm."*

Node polices *only* have a local (operating system) impact on how the local resources (e.g. CPUs, memory, local file quota) being used by jobs.

Relationship between Policies There is no relationship between three types of policies. They can be specified independently by different global entities and be checked at different levels of job execution and file accessing. Users supply policies to VO administrators so that their policies can be taken into account while VO resources are being allocated for their jobs. During a job submission, their policies will be checked and enforced alongside with the VO policies. Once the VO level policy check is passed, jobs are subjected to node level policies. Hence, in order to get access to a node, a job needs to satisfy the user's, the VO's and the node's policies. If a job fails any of these, it will not be executed on the node.

4 XtreemOS Security Architecture

In the previous chapter, we have presented the key concepts in an XtreemOS VO and discussed how to manage the global entities with the VO model. Based on this model, the present chapter describes a security architecture for supporting application execution and data management in XtreemOS.

This chapter consists of two parts. The first part presents the detailed design of the security services: **C**redential **D**istribution **A**uthority (**CDA**), **VO P**olicy **S**ervice (**VOPS**), and **NO**de-level **P**olicy **S**ervice (**NOPS**). CDA is a service responsible for delivering certified VO attributes to users in the form of a X.509 public key certificate, called a XOS-Cert. VOPS is a VO level policy management service to allow a VO admin and users to enforce their policies. NOPS is a node level policy management service to enable resource admins to enforce local policies on their machine.

The second part of this chapter presents an overall security architecture that has been designed to provide a secure operational environment for AEM and XtreemFS. We shall describe in detail how the security services are used to address the security challenges outlined in Section 2.2.1. Specifically, the presentation details how they are used to enable the *authentication* among global entities, and, that between global entities and AEM/XtreemFS services, and to facilitate coordinated *access control* in a VO.

4.1 Security Services

In the previous specification, five security services, Identity Service (IDS), Attribute Service (AttrS), VO Policy Service (VOPS), VO membership Service (X-VOMS), and Credential Distribution Service (CDA) are described to support secure VO management. However, not all these services expose interface to other non-security XtreemOS services *during application runtime*. For example, three security services, including IDS, AttrS, and X-VOMS services sit behind CDA to assist CDA to issue credentials to VO users. This section shall refine these security services

In this section, we shall focus on the detailed description of the global security services, which are defined as security services exposing external interface to non-security services in the XtreemOS system. They are:

- Credential Distribution Authority (CDA)
- VO Policy Service (VOPS)
- NOde Policy Service (NOPS)

The relationship between these services are shown Figure 4. CDA and VOPS offer interfaces to external (non-VOM) components, such as AEM and XtreemFS.

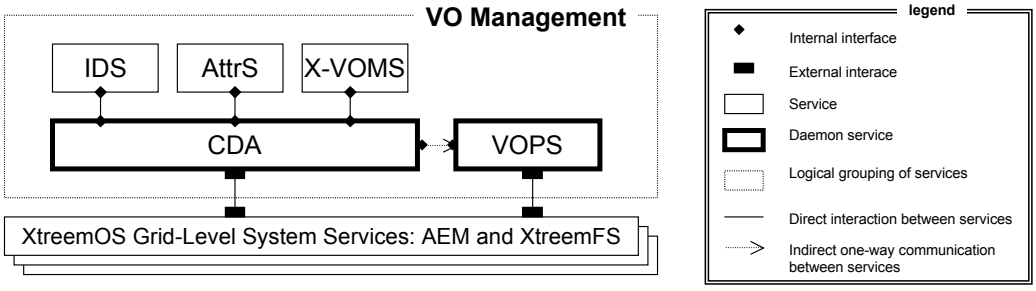

Figure 4: Relationship between Security Services

IDS, AttrS, and X-VOMS communicate with CDA through internal interfaces. There is an indirect one-way interaction from CDA to VOPS through a persistent credential store which contains a list of currently valid VO credentials for each *active* VO user. Thus, as well as enforcing VO policies for resource selection, VOPS can also enforce VO-wide resource usage control based on the information from the store.

4.1.1 Credential Distribution Authority

Motivations - Why dynamic distribution of credentials? Credential Distribution Authority (CDA) provides dynamic (on-the-fly) distribution of credentials (identity and attributes) to users so that

1. users *do not need* to rely on PKI to access XtreemOS services. Instead, users and applications, who do not have an established PKI infrastructure, can use their existing security technologies to access XtreemOS. Therefore, the difficulty of managing such an infrastructure is hidden away from end users and applications.

2. as credentials are dynamically distributed, our system can better cope with dynamism in a Grid environment, such as on-the-fly creation of new VOs, introduction and removal of users and resources from a VO.

Point (1) is appealing as existing applications need not to re-factor their code to make use of XtreemOS services. All they need to do is to configure their applications to access XtreemOS as the existing security infrastructure can be reused. From end user perspective, this is also useful as they do not need to be aware of new type of authentication methods and techniques. From security point

of view, the introduction of CDA can enhance the transparency of using Grids and reduce the complexity of integrating existing applications with Grids.

Point (2) is important for commercial applications that demand highly dynamic collaborations due to the nature of business interactions which are inherently dynamic. In contrast to existing static credential (mostly identity) distribution approach, our approach allows XtreemOS better flexibility and dynamism. It is flexible as this approach removes the need to force users to use certificates to access our system. As detailed in the first specification, a range of authentication methods to authenticate to XtreemOS will be supported. Most likely, users can rely on their existing security infrastructure to access our services without needing to master another type of authentication method. The philosophy is that without demanding additional security knowledge and expertise, it will be easier to allow existing users and computer systems to make use of Grid capabilities, at least from a security perspective.

Functional Description In XtreemOS, user credentials are embedded in a X.509 v.3 public key certificate. That is, a user's identity is the subject's DN field and attributes are extensions of a certificate. As such certificates contain XtreemOS specific extensions (such as GVID, GUID, GGID, see Section 3.3), we call them XOS-Cert. XOS-Cred is the pair of an XOS-Cert with the corresponding private key.

CDA can issue credentials to all three Global entities: VOs, users, resources. When CDA issues credentials to VO resources, it is also referred as Resource Certification Authority (RCA). As its critical nature, CDA has to be boostrapped from a trusted source and runs in a trusted environment.

CDA uses the private key of the VO manager to sign users' and resources' certificate. When more than one GVIDs are included in the extension fields of a certificate, the private key of the *first* VO manager, if there are more than one VO managers involved, will be used to sign the certificate.

As mentioned at the beginning, CDA relies on three other security services, IDS, AttrS, and X-VOMS, to issue credentials. In the remaining of this section, we shall take a closer look at them.

Identity Service Identity service serves two purposes:

- generate and allocate global unique IDs: GVIDs, GUIDs, and GGIDs.
- provide an interface allowing integration of third-party identity infrastructure (e.g. Shibboleth) into XtreemOS

There are many ways to ensure the global uniqueness of an identity, be it numeric or alphanumeric. For some of these mechanisms, please, for example,

see [13]. Some of candidate approaches rely on the availability of a global (across all CDAs) online directory service to check the uniqueness of IDs. We tend to favor on those approaches who can provide uniqueness assurance without online interactions (as they scale better), although this may mean some compromise on the quality of the IDs in the sense that IDs generated could have a small probability to clash.

Validity of an identity In contrast to the short-liveness of the users' certificates, VO and resource certificates should have longer validity period.

Usage of identity In XtreemOS, IDs are used in two different ways:

- The identity of an entity is checked by authentication protocols to verify the authenticity of an entity.
- The global IDs are used by nodes to map into local UID/GIDs to provide user account level isolation.

Attribute Service The purpose of attribute service is to provide global entities with attributes. Section 3.3 has presented attributes for VO users. In XtreemOS, they are used:

- to assist identity service to generate GGIDs as for different combinations of attributes,
- to allow nodes to perform access control to their resources based on non-global attributes; and
- to allow nodes to generate local OS level GIDs.

Membership Service XtreemOS Virtual Organization Membership Service (X-VOMS) provides VO membership checking service that would allows systems to validate the VO membership of a user who initiates a Grid request from a Linux terminal. X-VOMS connects to a database which stores VO information, such as identity and attributes, about a user.

Interfaces X-VOMS has two interfaces: management interface and online checking interface. The management interface that allows VO administrator to add, remove, and modify users and their information. The online checking interface acts as a backend of the CDA service for checking users' VO membership, global IDs, and attributes. This procedure is to ensure the credentials issued by CDA is authentic and valid in the up-to-date membership database.

4.1.2 VO Policy Service

VO Policy Service (VOPS) mainly provides policy decision points to VO administrator so that VO level access control can not only be enforced at nodes but also at VO level. The aims is to ensure coordinated resource usage, including sharing *and* isolation.

One benefit of integrating policy decisions at the VO level is to accommodate the flexibility of incorporating VO policies in job scheduling and resource negotiation processes. Additional functionalities of VOPS are policy administration and information points.

Having multiple policy decision points in various stage of job execution can be complex. For example, when VO policies and node policies are processed together, it is possible to have policy conflicts. Therefore, in order to minimize the impact of such complexity, VOPS is designed to be independent from node policy decision points.

Together with node level policy decision points, VOPS forms a hierarchical access control framework that can be tuned to achieve various degrees of control to resource usage within a VO. It is being used by AEM to facilitate VO policy governed resource selection and job scheduling. It can also be used together with the accounting service to enforce constrains (e.g. quota and usage pattern) to certain types of resource consumption in a real-time manner.

The D3.5.3 defined following AEM use cases: Job Creation, Resource Matching, Resource Negotiation, and Job Execution which are described to detail in the section 5.4 of the mentioned document. There is no need to redefine all the AEM related use cases because there have not been any changes to the AEM services, which are managing the described processes. However, we shall focus on the AEM and VOPS interactions because, currently, VOPS only serves AEM job execution management.

The main process handled by the Application execution manager is job execution, which includes job submission, resource discovery and allocation, and control over the execution. In the following, we describe two major AEM processes: resource discovery and allocation, as they both relate to the VOPS.

Resource discovery This process consists of finding a set of suitable resources, which satisfy the resource requirements described for the job. At first a potential list is retrieved, which is later on refined based on actual resource state (dynamic properties). This second step has to include the check with the VOPS in order to guaranty, that all the selected resources are allowed to run a job for the user according to the VO policies. Hence, the functionality of VOPS is to act as PDP to enforce VO policies on resource usage based on the user and resource properties.

Resource allocation After the job has a list of candidate resources, which satisfy the job requirements and are compliant with VO policy, the job manager contact the local resource allocation manager on a remote node in order to negotiate job execution. In the negotiation process, the allocation manager should check that the request is valid (whether it is signed by the VOPS), and that it is not in conflict with the local resource policies. The VOPS is not directly involved in this step but the allocation service can check the VOPS signature in order to confirm the legitimacy of the requests.

4.1.3 Node-level Policy Service

VO-level Policies versus Node-level Policies

VO-level policies specify whether grid users are able to access one or multiple node(s) that participate in a VO. Resource owners (a.k.a. domain administrators) have their final control of external VO access to local nodes by specifying Node-Level Policies (NLPs). NLPs may comprise *access control policies* (to specify what kind of access permissions that VO users could have upon a set of resources) and *resource usage policies* (to specify what kind of constraints or throttling policies under which resources are consumed by VO users).

For a VO user, the successful launching of a job on a resource node has to pass the checking of both VO-level policies (managed by VO admin*s*) and node-level policies (managed by local node admins or domain admins). Generally the *least privilege* principle should be applied when there are any conflicts between polices at these two levels.

Application-independent versus Application-specific

NLPs can be application-independent or application-specific. Application-independent policies are those regulations on the usage of Operating System(OS) objects such as CPU, memory, disks, files, and so on. These policies could be enforced with the direct support from local OS. Application specific policies are those regulations on the usage of objects managed by specific application services such as web services, database services and batch job management services. Application-specific policies are expressed in different manners (e.g. in different formats of configuration files). For example, a MySQL database server could exert a size limitation of 8Gb for each database created by a VO user, and a local batch job manager may restrict that a given VO user can only submit jobs on a specific queue with some predefined priorities. The enforcement of application-specific policies is performed by local application services .

Functionalities

Besides the coordination with high-level security services like VOPS, basic functionalities of node-level policy management are identified as follows:

Build-in support of policy enforcement on OS objects To facilitate the secure and efficient sharing of nodes among VO users, it is necessary to provide build-in support of policy enforcement of application-independent OS-aware objects. These objects could be categorized into: CPU, memory, files (including devices) and network.

Access control policies Access control policies are mainly related to files (here we only consider local files, while the access control of global files is handled by *XtreemFS*). Besides file permission bits, it is nature to use Access Control Lists (ACLs) to specify more fine-grained file access policies. Sockets could also be considered as files. For example, a VO user could be granted with *read*, *write* or *listen* permissions on a specific range of ports.

To perform the enforcement of access control policies on the node, VO users must be mapped onto local Linux user accounts at first. During the mapping process, access control policies are populated into local security configurations such as file permission bits and ACLs, which is bundled with the mapped accounts (and possible local groups) if necessary. For advanced fine-grained access control on OS objects, security enhancement mechanisms like SELinux could be leveraged.

The general syntax to describe an access control rule is as follows:

```
<subject> <object> <permissions>
```

where `subject` is a pattern for matching VO users (or groups, subgroups, roles), `object` is a pattern for matching OS objects (e.g. *files*, *directories*) and `permissions` are a composition of access rights (*read* / *write* / *execute* ...) that `subject` could have over the `object` .

Resource usage policies Resource usage policies are mainly related to quotas, such as CPU time limit, memory limit, disk quota and network bandwidth throttling. Quotas may be *absolute* values or *relative* ones. For example, scientific computing applications are more concerned about the absolute quota of cpu usage (e.g. a job's max running time is limited to 1000 cpu hours) whereas commercial applications like web hosting ones put more focus on relative quota of cpu usage (e.g. a virtual website can process incoming requests by taking less than 60% of CPU load of the host machine).

The enforcement of resource usage policies could be partially done via the system call *setrlimit()*. For the enforcement of relative quota on resource usage, more advanced kernel support is required. Fortunately, new isolation and virtualization mechanisms like process containers, KVM, openvz, Xen are available for use, which provide extensive support for system resource partition.

Similar to an access control rule, the general syntax to describe a resource usage rule is as follows:

```
<subject> <object> <quota>
```

where `subject` and `object` have the same meanings as above, `quota` is an expression to specify the usage policy for `subject-object` pair in the same line. Besides quota values, there could be other auxiliary information in the `quota` expression, such as time frames when the rule is applied or not applied.

Integrated support of policy enforcement on application-specific objects To define a generic abstraction of all kinds of application-specific policies is beyond the scope of current version of security services. It is also not realistic for security services to take over the control of all kinds of policy enforcement. Whereas it is necessary to allow local admins to express application-specific policies against VO users, and integrate local application-specific policy engines such as the privilege subsystem of a DBMS or scheduling components of a batch job manager for policy enforcement.

Nowadays either open-source or commercial job manager software (e.g. PBS, OAR, Condor, LSF, Maui, etc.) provide very flexible and effective scheduling policy support for a cluster, though they adopt different proprietary languages to express the policies. Moreover, applications like SAP Web Application Server use their own set of security scheme for authentication and authorization. A coarse but flexible way to link with these software and take advantage of their policy support is *scripting drivers*. The syntax of an application-specific policy rule is as follows:

```
<subject> <app-policy-driver> <arguments>
```

where `subject` has the same meaning as above, `app-policy-driver` is the extern callout to establish the specific policy configurations required by applications. `Arguments` are used to pass additional information that only recognized by `app-policy-driver`. When the callout is launched, the mapped local account information corresponding to `subject` will be passed into it.

4.2 The Security Architecture

To achieve clarity, we present the security architecture from two angles: one is from the perspective of AEM and the other from XtreemFS. We describe how the security services are used to enable three security mechanisms (authentication, authorization, and delegation) based on their functional description outlined in Section 2.1.

4.2.1 Securing AEM

This section describe a security architecture for supporting application execution management in XtreemOS. This is illustrated in Figure 5. To assist the understanding of how the security services provide its support, this figure deliberately follows the same presentation style as the AEM functional description figure (i.e. Figure 1 of this document) presented in Section 2.1.

In Figure 5, the client side is represented by the XtreemOS console, which is (currently) a command line interface to allow users to type in AEM commands, such as *Xsub* for job submission, and *Xps* for querying current jobs.

XOSD is a server-side daemon service running on *each* XtreemOS node for managing jobs (e.g. submission, execution, monitoring), and resources (e.g. allocation and negotiation). In Figure 5, it is shown running on a client node as *XOSDc* and a resource node as *XOSDr*. XtreemOS console and XOSD are node (level) services.

AEM has two distributed services (classified as external services in the figure): JobDirectory and ResMat (ResourceMatching). JobDirectory provides job query services to users to retrieve the current state of their jobs. ResMat is responsible for finding resources (machines) that can satisfy the criteria in the job specification and have spare resources (e.g. CPU wall time) to provide at the time of job submission.

All security services (CDA, VOPS, and NOPS) are involved in the security architecture to support AEM job submission. Apart from NOPS, which resides on each resource node as node services, all others are global (VO-wide) services. Figure 5 also includes the additional steps describing the interactions between the security services and AEM services.

Configuration of the System All global services run on machines separated from the client and resource nodes. During the setup stage of the system, the following requirements should be satisfied.

1. The user has registered with the VO manager of the VO - esvo. Thus, according to Section 3.3.4, this implies that all the identifiers and attributes are allocated and the VO databases have been properly populated.

2. The user has established a means to authenticate with the CDA. Such means can be one of the three authentication methods (username/password, public key certificate, or Kerberos), as specified in the previous deliverable [5].
3. The VO admin has generated two proxy certificates based on the VO manager's public-private key pair, one each for CDA and VOPS. The private keys and the corresponding proxy certificates are then installed on the following security services respectively:
   (a) CDA
   (b) VOPS
4. The following services have access (**either via online access or through offline installation**) to an *authentic* copy of the VO manager's public key certificate and the CDA's proxy certificate:
   (a) JobDirectory
   (b) ResMat (i.e. ResMatching)
5. The PAM module and the XOSDr has access to an *authentic* copy of the VO manager's public key certificate (**e.g. through offline installation to the node's trust anchor by the resource admin**) , and, the proxy certificate of CDA *and* VOPS.
6. The followings need to have a genuine copy of the resource node's issuing CA's public key certificate:
   (a) The client node
7. There is an established means to authenticate the VOPS and ResMat services with each other.
8. The system on the resource node has properly configured XtreemOS **P**luggable **A**uthentication **M**odule and **N**ame **S**ervice **S**witch installed. *Note that all the PAM and NSS modules are provided by WP2.1 and are specified in the deliverable [2].*

Interactions within the Security Architecture

1. A user types the *Xsub* command to the XtreemOS console. A simplified example is shown below:

   ```
   Xsub -vo esvo -group testing -role programmer -f job.jsdl
   ```

   It means that the user who is a programmer in the testing group wants to submit a job, with the job specification job.jsdl, to a VO, called esvo.

   1.1 XOSDc requests a XOS-Cert from the CDA service via an established authentication means.

1.2 XOSDc receives the XOS-Cert securely.
1.3 XOSDc verifies the integrity of the XOS-Cert by using the (pre-installed) VO manager's certificate.

2. XOSDc generates a jobID and creates a job with the jobID.
3. XOSDc registers the job to the JobDirectory with jobID, and the JobMng address with the user's identifier and XOS-Cert.

3.1 Upon receiving a job registration request from a user, the JobDirectory authenticates the user based on the XOS-Cert

4. registration: success or failure
5. once the job has been registered, XOSDc starts scheduling the job by contacting ResMat with a list of job requirements (i.e. job.jsdl)

5.1 The ResMatching authenticates the user based on the user's XOS-Cert.
5.2 Once the user is authenticated, the ResMat forms a policy request based on: (a) the list of job requirements; (b) the list of candidate resource nodes belongs to the VO - esvo from the Application Directory Service (provided by WP3.2 - highly available and scalable services); and (c) the description of the user (e.g. the identifiers and attributes extracted from the XOS-Cert). Subsequently, this XACML policy request is forwarded to the VOPS.
5.3 The VOPS checks the request against the VO policies and signs the decisions using the private key (of the VO manager's proxy certificate).
5.4 The VOPS sends back a signed policy decision to ResMat.

6. ResMatching returns a list of authorized (i.e. signed decisions from VOPS) resource nodes to XOSDc.
7. XOSDc performs a resource selection algorithm to decide which (set of) nodes to be used.
8. XOSDc negotiates resources with XOSDr on the selected nodes until all the required resources (e.g CPU, memory, queue ...) are reserved.

8.1 Before the negotiation begins, the client and resource nodes perform mutual authentication to check the authenticity of each other.
8.2 XOSDr checks whether the access is authorized by the VOPS by checking the VOPS's signature
8.3 Once the authentication and authorization checks are passed and the negotiation completes (so that the resource node is clear what is required by the client node), the resource node checks its local policies using the NOPS.

9. The reservation confirmation is then sent back from XOSDr to XOSDc.

10. Once all resources are reserved, XOSDc submits the job to the corresponding (set of) XOSDr, who will first check whether this is a job submission with an existing resource reservation.
    10.1 If it is with an existing reservation, the PAM generates a proxy certificate [13] and a corresponding private key for the user. This certificate, *together with the private key*, is called XOS-Cred.
    10.2 The PAM modules allocates UID/GIDs according to the attributes embedded in the XOS-Cert.
    10.3 The mapping between the local UID and the GUID and that between the user's attributes (e.g. group and role) and local GIDs are stored in the account mapping database on the resource node by the account mapping service developed by WP2.1.
    10.4 At the meantime, the user's XOS-Cred, is stored in the kernel key retention service by the PAM.
11. XOSDr forks a user process for the job.
    11.1 The system sets the UID and GIDs for this process. The job will then start its execution on the node.
12. The confirmation of job submission is sent back to XOSDc. This completes the job submission process.

4.2.2 Securing XtreemFS

This section describe a security architecture for supporting secure data management in XtreemOS. This is illustrated in Figure 6. To assist the understanding of how the security services provide its support, this figure deliberately follows the same presentation style as the XtreemFS functional description figure (i.e. Figure 2 of this document) presented in Section 2.1.

A number of additional services have been added to the original functional overview diagram to provide a secure data accessing environment in XtreemOS. They are:

- CDA, which provides the XOS-Cred
- the key retention service, which stores the XOS-Cred
- the account mapping database, which stores the mapping between the local UID and the global GUID and other global credentials
- PAM/NSS modules, which stores the mappings into the account mapping database

In the figure, two interactions are highlighted in dashed lines because they occur *before the user process is forked by the system*. These interactions are:

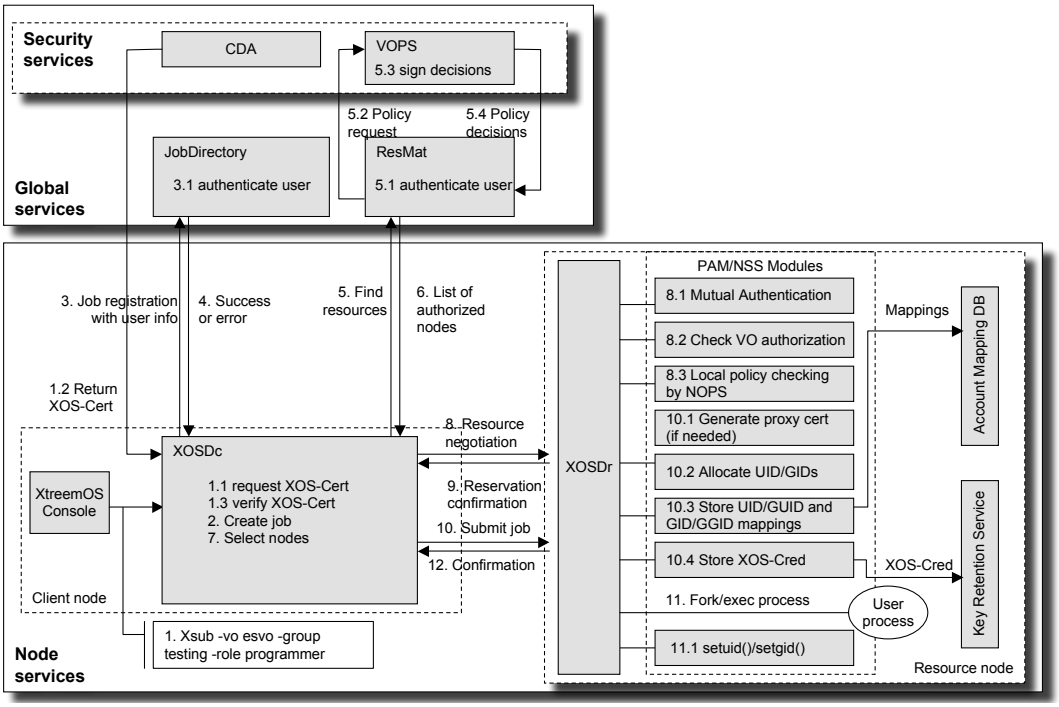

Figure 5: A Secure Application Execution Environment in XtreemOS

- the interaction between the CDA and the key retention service, and
- the interaction between the PAM/NSS modules and the account mapping database.

Basically, there are two typical occasions that the system can spawn a new process based on the service provided by the PAM/NSS modules. In the first case, such a process can be created by the AEM XOSD on a resource node, just as described in Section 4.2.1. In this case, the XOS-Cert of the XOS-Cred is a *proxy certificate* generated by the system, as explained in that section. As the result of a successful job submission, the mappings are stored in the local account mapping database and the XOS-Cred is associated with the user in the kernel key retention service.

The second case is that the user explicitly logins into a VO, and thereby, the system spawns a new process for the user with the UID/GID supplied by the PAM module. The outcome is similar to case one. The only difference is that the XOS-Cert is an *end entity certificate* issued by the CDA.

Configuration of the System During the setup stage of the system, the following requirements should be satisfied.

- The MRC has an authentic copy of the VO manager's public key certificate installed.
- The node and each OSD has access to an authentic copy of the MRC's public key certificate installed.

Interactions within the Security Architecture Figure 6 illustrates the interactions among MRC, OSD, XFS client (aka. XFSc) and a detailed list of additional interactions related to security.

1. A file (operation) request (e.g. create, open, delete, read, write a file), initiated by a user (application) process, comes into the XFSc via the Linux kernel[9].
   - 1.1 The XFSc requests local credentials from the FUSE daemon by passing in the PID of the calling process
   - 1.2 The XFSc receives UID/GIDs back
   - 1.3 The XFSc requests global credentials by UID from the accounting mapping service (developed by WP2.1)
   - 1.4 The XFSc receives GUID and a list of other global credentials (e.g. GVIDs) from the accounting mapping service
   - 1.5 The XFSc fetches a XOS-Cred from the key retention service for this GUID
   - 1.6 The XFSc receives the corresponding XOS-Cred
2. With the global credentials, the XFSc asks the MRC whether the operation is permitted.
   - 2.1 The MRC performs a mutual authentication protocol with the XFSc, who uses the XOS-Cred to authenticate with the MRC.
3. If the operation is allowed, the MRC returns with a set of capabilities, which is signed by the MRC, and a signed security token for *this specific user* to allow him to access the corresponding data (The security token should be tided to a specific user and a specific piece of data.) Otherwise, the operation is deemed as denied.
   - 3.1 the XFSc verifies the integrity of the capabilities.
4. With the capabilities and the token, the XFSc contacts the appropriate OSD(s). (Only one OSD is contacted in the diagram to illustrate the scenario. But in reality, a file may be stored in multiple OSDs.)

[9]This kernel must be compiled with the Filesystems in Userspace (FUSE) kernel module and support the Virtual File System. See the deliverable D3.4.1 [1] and D3.4.2 [7] for more details

4.1 the OSD checks the authenticity of the token (to make sure that it is from the MRC) and the integrity of the capabilities

4.2 if both are satisfied, the OSD performs a mutual authentication protocol with the XFSc.

5. The OSD(s) transfers the file to the XFSc. If this process fails, errors will be reported.

   5.1 the XFSc checks the integrity of the file.

6. If the file passes the integrity check, XFSc presents the file back to the user process.

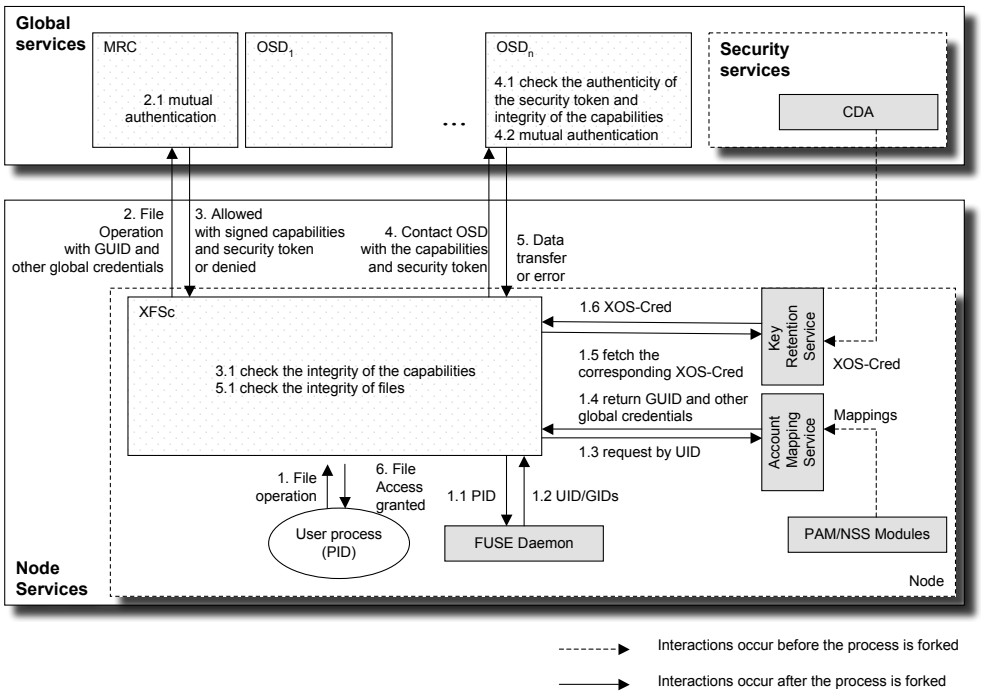

Figure 6: A Secure Data Access Environment in XtreemOS

4.3 Discussion: Ongoing Security Challenges from Other WPs

At the time of writing, XtreemOS is still a very active project. Because of this, we perceive it is likely that our design will need to be adjusted to suit the new security requirements that may emerge in the continuously improved design come up by

other WPs. Two potential candidates of such, that have just started to emerge in the project, can be drawn from AEM and XtreemFS.

As shown in the architecture document [6], the ResMatching component has been removed from the AEM architectural diagram and a new AEM component, called Reservation Manager has been introduced. This is due to the fact that some design decisions between WP3.3 (AEM) and WP3.2 (highly scalable and available services) have been made. Similarly, although in the security architecture for XtreemFS, the secure association of capabilities to a user is generally referred as signed capabilities with security tokens for the user. As stated in D3.4.1 [1] and our communication with WP3.4, the exact security protocol that is needed to facilitate the interactions is not yet decided.

However, we plan to continue the use of PKI as the underlying security infrastructure to support the overall security architecture because of its well-known scalability properties in a large scale distributed computing environment.

5 Technical Discussion

This technical discussion of the XtreemOS security specification is divided up in two parts: (1) an architectural discussion and (2) a discussion of the proposed specification from a business application perspective. The latter area has been considered as it presents different requirements for security than the scientific applications that have traditionally motivated Grid computing. The reason for including this technical discussion, towards the end of the document, is mainly to clarify the design decisions made within the specification. Secondly, it assess the scope and capability of the specification and provides some guidance for developers and administrators that will need to manage the specification either during implementation or runtime. Thirdly, the technical discussion explains the scope of the specification and identifies some additional requirements that we have given thought to, even though they might not be currently fully addressed in the specification. This is an ongoing process again with the intention of guiding developers, administrators and users in extending or configuring the security mechanisms appropriately, or identify further work to be done to assure XtreemOS for use in a secure commercial production environment. The details and implications of these requirements are not yet fully worked out but nonetheless are discussed in the two sections on "Legacy Application Runtime and Management" and on "Strong Isolation".

By identifying the additional requirements which is an ongoing process, we are able to either guide developers, administrators and users in extending or configuring the security mechanisms appropriately, or identify further work to be done to assure XtreemOS for use in a secure commercial production environment. The details and implications of these requirements are not yet fully worked out but nonetheless are discussed in the section on "Legacy Application Runtime and Management". These requirements are new requirements being studied and examined in parallel to the work presented in this deliverable. As they are new findings in the project, they are also been described (from both security and non-security angles) in a parallel deliverable produced by WP 4.2 in D4.2.4 [4].

In addition, it must be noted that XtreemOS software will be released as open-source, such that there is (1) no hiding of how security mechanisms are implemented but (2) possibility for skilled developers to alter and redeploy code in order to gain advantages. Alternatively, they might be able to study vulnerabilities in a more complete manner, as they can perform white-box analysis on the code base.

5.1 Architectural Discussion

Scalability, flexibility, autonomy and decentralization are four architectural aspects that need to be considered in XtreemOS, due to the goal of supporting largescale and potentially cross-organization systems. We do not only focus on the basic security goals of authentication, authorization, confidentiality, integrity and isolation emphasized in the first specification. These have been compiled as a set of security architecture principles that will be used to rate the current architectural specification, as well as guide further iterations and configuration constraints. These architectural principles have been derived by considering requirements from both business and scientific applications and the management of the infrastructure supporting them. They are however grouped under the 4 headings introduced above.

5.1.1 Scalability

One of the major reasons for moving Grid mechanisms into the OS as opposed to at the middleware level is to remove some of the overheads associated with going through several layers of software. Secondly, largescale applications look to Grid-solutions as a means of increasing performance and availability. Therefore, the security mechanisms introduced by XtreemOS should not act against these specific objectives. There are three scalability principles that we have derived for XtreemOS security.

Fast membership and group lookups and decisions For applications with very high response time requirements yet that need to validate the affiliations of principals on a per-asynchronous-message basis, it is necessary that the bottlenecks introduced by membership and group checks be minimised. The specification has not yet matured to the level where we can properly assess these points. Implementations may have to be tweaked to include caching and localised distributions of membership lists in order to stay within the response thresholds demanded.

Minimise expensive Remote Procedure Calls (RPCs) The overhead of RPC calls depends on the type of transport protocol and the transmitted data size. As it is impossible to totally eliminate RPCs in a distributed system, we would like to find means of minimising them, avoiding synchronous calls where necessary and avoiding bulky data packet sizes unless delivering content. Currently interactions with the VOPS to check VO-level policies appears to be a point of concern for high RPC, such that we will need to focus on techniques for more effective policy

distribution and, to some extent, reliance on localised checks. Caching mechanims may also be introduced, but one needs to be aware of the tradeoffs between performance, data-provenance and privacy that will arise here.

Avoid redundant security protocol actions, certificates and keys The overhead introduced by public key encryption is negligible these days with the increased capability of processors and more efficient protocols being introduced. However, in order to support scenarios that introduce several concurrent users in different VOs, the strain on the various resource providers might be too high, if they need to perform too many expensive certificate validations, store an extensive list of public key certificates and continuously monitor their lifetime. However, as noted in the specification of credential services, we avoid the constraint that users need to rely on a public key infrastructure to access XtreemOS services. This also adds to the flexibility objectives of the architecture.

5.1.2 Flexibility

The XtreemOS architecture must remain flexible in order to support various organizations, applications, policies and types of users. The flexibility goals are therefore with regards to how resources are shared, selected, accessed and configured. Most of the flexibility goals for XtreemOS security are dependent on the generality of the VOPS service and its policies. This however may lead to some issues regarding manageability, which we also should not ignore.

Flexible sharing and multiple group membership Users and resources will have several different responsibilities in any physical or virtual organization. They will need to be able to properly coordinate these without violating policies by disabling security mechanisms and exhausting resources. There is then a need for users and resources to be in multiple groups with different privileges at the same time. Fine-grained, role and action-based access controls should also be possible with the types of policies specified. The XOS-Cert enables the inclusion of multiple group names in the extensions. However, we do not yet specify how to selectively include credentials and relevant group names in requests exchanged between different resources and users.

Selective storage of data Although XtreemFS offers a solution for wide-scale distributed storage of data, it should still be possible for organizations and users to place restrictions on where their data is stored. That is, users of XtreemOS nodes may mount XtreemFS as their sole filesystem, as they are satisfied with the

scalability it offers. However the risk of leaking sensitive information often outweighs the adoption of distributed storage mechanisms. The current solution in XtreemOS to this problem is to allow users to specify VO policies in the VOPS concerning restricted storage of files. However, this introduces a scalability flaw, as blacklists may need to be maintained per user and possibly per resource i.e. resource providers may want to block certain owners of data from storing on their machines. We may want to further introduce some form of mandatory security into the system that requires labeling of data, but this would again have implications for the manageability.

Selective levels of isolation Isolation of application processes and data is a significant requirement and principle for robust security in resource-sharing systems. We have introduced isolation in the previous specification but, after internal discussions, found that this cannot be a set-in-stone mechanism. It should be possible for users and resource providers to turn on and off isolation as it has some performance penalties that might even affect scalability. Moreover, we wish to emphasize the idea of different levels of isolation (including 0), which change the way in which nodes and groups of nodes are configured. This feature is not yet included in the specification but should be possible with a form of local, node-level policy enforcement.

Resource-respecting security mechanism selection Following from the above, node-level policies should be capable of being used in order to negotiate which cryptographic suites are used or if none at all. The decoupling of credentials, identities, attributes and secure communication help to enable this flexible selection. This is particularly important for the mobile versions of XtreemOS that will need to be considerate of the resource constraints of these devices.

5.1.3 Autonomy

Administrator-independent access control, resource and group management This principle comes from the autonomy goal of Grid systems and general shared resource environments. The expectation is that basic participants (without special administrative privileges or responsibilities) in a shared resource environment should be able to define access controls to resources they own, as well as create and edit the membership of groups that have access to their resources. This should be possible even in a distributed, cross-organization setting, where scenarios such as shared projects and business processes are envisioned. This is one area where the XtreemOS security architecture will need to pay more attention as we move towards addressing implementation and management concerns. According to the

current specification, it is still assumed that VO managers and owners delegate these tasks to dedicated VO administrators.

Support cross-domain sharing with minimal disruption to local native accounts Although today's organizations encourage collaboration with external partners and across corporate borders, there is resistance to changing internal settings and policies for the sake of collaboration. The impact of collaboration and networking resources in a shared manner should therefore be properly controlled, such that local administration, distribution of privileges and group management should not be hindered or changed for the sake of sharing - unless there is an extreme case. The XtreemOS solution to this is based on account mapping as opposed to demanding changes in local user and group names. Maintaining these mappings however introduces a new management task, which we continue to improve.

Support for nested subgroups Creators of groups should be able to further restrict the membership of these groups easily, such that privileges can be inherited. This is currently not well-specified in the current specification, as it is an optional and low priority requirements, but can be treated as an extension to the certificates.

5.1.4 Decentralization

Decoupling of identity, group membership and authentication mechanisms The decoupling of these is important in a distributed, cross-organizational system, as the way in which different types of entities are issued identities, assigned to groups and authenticated will change according to their technical properties and organizational affiliations. This is fully satisfied by XtreemOS with the clear separation of the Credential Distribution Authority (CDA) from the Identity Distribution Service (IDS).

Decentralized authentication, group membership management and authorisation For the purpose of scalability, flexibility and privacy, it should not be necessary that a network of XtreemOS nodes be restricted to one central point for authentication and managing membership in VOs or groups. In addition, there should be no shared user database across organizations for maintaining identities, as this would force organizations to expose details of their internal policies and personnel. This is currently unclear as we do not explicitly define how the security architecture works for cases where there are multiple VO managers.

5.2 Legacy Application Runtime and Management

Legacy and large-scale business applications are of particular interest for evaluating the XtreemOS security architecture, as they bring requirements that are not typically associated with Grid applications. Legacy Applications are understood to be those not initially designed for execution in a Grid environment and, moreover, not previously deployed on top of a distributed operating system platform such as XtreemOS. Supporting legacy applications includes the scripts, batch files, installers and runtime of the application, each of which have been implemented and compiled with some assumptions of the underlying OS. One of the key problem areas is that of identity management, user management and group management. For this reason, the emphasis in this section is placed on these aspects of deploying and executing systems. The main concern here is that deploying and configuring legacy applications which are built on the traditional POSIX understanding of users, groups and ACLs will either fail at start up (as a result of non-compliance) or may grant/deny access to the wrong users, if the identity is incorrectly interpreted, truncated or formed. POSIX uses numeric integers known as User Identifiers (UIDs) and Group Identifiers (GIDs), and are based on a flat namespace without any notion of a domain identifier. However, in a more global identifiers required in a distributed, cross-organization system will need to maintain identifiers for domains and hierarchies for the sake of cross-organization uniqueness, autonomy and scalability. This is a classical problem for porting legacy applications across platforms (i.e. if the identity concepts are incompatible), as well as deploying them across a distributed OS. Note that although these potential problems are emphasized within the security specification, these incompatibilities could have wider implications e.g. job management, scheduling, execution etc. The goal is hence to show the scope for the current specification in solving this problem and to identify technical directions for future extension and application of the specification.

5.2.1 The Application Installation Process

This section describes the application installation process using an adaptation of the Application Execution Management (AEM) Figure 1. The adaptation is illustrated in Figure 7 and explained first below. The scenario considered is that of a distributed data center, where an administrator needs to quickly deploy and make available a hosted application for a customer. One goal is that the installer (which is essentially a script interpreter) should remain unaware of the distributed OS platform and execute as if it is on one central machine.

1. The administrator starts the application installer, which is located on some machine in the data center's administration;

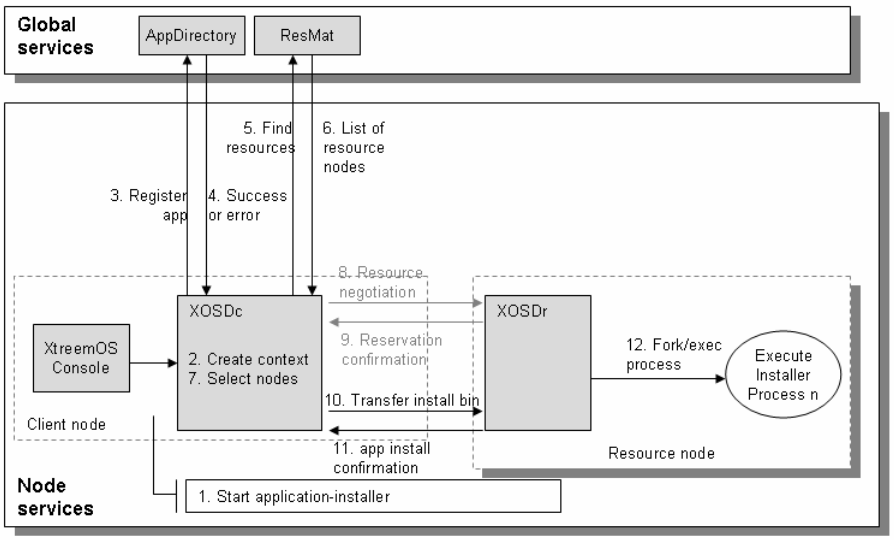

Figure 7: The application installation process

2. The XtreemOS client daemon receives the request, creates an application context for the installer and

3. registers the application with a central application directory (built on the Job Directory) mainly for accounting purposes;

4. If the application can be registered (e.g. if the customer is still active) then a success is returned;

5. The OS daemon then needs to find resources available for installing and hosting the application; for a typical application server this includes for the database, different server containers and various server-side software components.

6. A listing of available resource nodes should then be returned and

7. a selection process is continued, which should take into account the customer's deployment policies.

8. Selected nodes are negotiated with for resources, again invisible to the installer (therefore it is assumed that the installer does not time out) and

9. a reservation ACK is returned once the target node is ready. All nodes should then by this time be capable of resolving identities within a common namespace that is unique for the particular application context.

10. The binaries packed with the installer are then transferred to the respective target nodes and

11. a confirmation of starting is returned - this should then update the installer's status indicator.

12. Finally, and most critically, each installation binary now executes on the given node, given that it has the correct privileges to complete.

The challenge set here is that there should be no changes necessary in the installation script, as this today executes successfully on a large Linux server. The problem is however that large Linux servers are not always available and it is desirable that one large server could securely host and install multiple applications of different customers. For step 12, it should be noted that the targeted server could already be hosting multiple database instances, web servers and components that belong to other customers. In principle, the security of XtreemOS should allow this installation without interference yet without having to change the basic installation script of the application i.e. the transparency principle of XtreemOS. The steps here are referenced in order to identify particular problems, discuss how XtreemOS solves these problems and, in cases where there are some technical challenges remaining, explains some alternative for getting around these with the current specification.

5.2.2 User Management

One of the challenges for XtreemOS, including security, is to support both Grid and Legacy applications. If an initial decision was made to support only one, then user management would not pose new complexities. User management for an application installation is understood as the following seven (6) aspects:

1. *Identification and registration*: associating an operating system user with a unique, repeatable, unforgeable identifier

2. *Role/Group assignment*: assigning the user to a specific role or group that gives them certain privileges for installation

3. *Customization*: having the OS and applications appear and behave in a way that is parameterized by the user (e.g. sizing of the machine)

4. *Authentication*: providing mechanisms that can associate all executables and messages with a particular principal

5. *Authorization*: providing mechanisms that ensure that only processes belonging to an authorized user can access objects, applications, data and custom settings for installation

6. *Accounting*: providing mechanisms that associate resource usage with an individual user or the customer on whose behalf they are acting

The critical point is that of identification and registration. If this fails, then the other user management operations will also fail. In XtreemOS, identification and registration is done using credentials specified in the XOS-Cert, which carries attributes that refer to the user's unique identifier and groups across the data center. Thinking in terms of group assignment and authorization, if privileged access to a resource is required, such as is often the case during installation of an application (especially for purposes of sizing the target machine), then the current application context within which the deployment or installation process is running must be part of a privileged group, otherwise:

1. An application installation procedure is not able to perform privileged actions in order to configure the operating system to perform best according to the application's custom requirements.

2. An application installation procedure is not able to generate additional users or groups.

The two above problems may occur as early as in Figure 7, step 1 and, if not resolvable, again in step 12, causing the installation to fail (taking into account that today's large installations can take a minimum of 2 hours to be completed, several failures are not tolerable). One technical precaution that needs to be recognized is the need for an application's scripts or installers to run under multiple user accounts, belonging to different groups and with special privileges. XtreemOS support this using either a generated or manual approach of creating a certificate with the required attributes; these attributes are the GlobalUserID, GlobalPrimaryGroupName and optionally GlobalSecondaryGroupNames. On login by the installer of the application, their client is issued a certificate with the relevant attributes. However, they would need to re-login in order to activate the new user, unless the assumption of roles or a persistent user pool was included. If there is a need to re-login, this would cause a break in the automation offered by legacy installation scripts, as an installer, which relies on the changes in the user or group database would not continue. Furthermore, this would not scale for multiple applications on nodes belonging to the same VO. It should also not be dependent on the application to dictate the naming convention for the mapping the GlobalUserID to the local UID, as this may result in mapping conflicts.

It is possible to by-pass these problems by introducing a constantly active superuser on nodes just for the purpose of installation, such that any global credential, belonging to an authorized installer, would be resolved to the same, privileged user. This however is opening up a point that, if exploited, could have wide-spread consequences. For this reason, carefully managed groups is an additional area that is significant for legacy applications.

5.2.3 Group Management

Group management in XtreemOS is also challenged by the global-to-local domain mapping problem identified above. In a data center, the validity of global groups suggests that it is always possible to resolve these to local, numeric group identifiers (GIDs) and privileges on all nodes in the data center. For the case of application installation, the questions are therefore, recalling that applications for multiple customer domains and namespaces may be concurrently executed on the target node:

1. where do the group names come from?
2. when are they created?
3. how are the privileges for groups created?
4. how are the user accounts of the installer added to the necessary groups?
5. how and when are the global group names resolved to local POSIX GIDs?

The group names must be valid, established and resolvable from step 1. In the worst case, consider a scenario where this is the first time that the particular application is to be installed. The installation's administrator must have a valid XOS-Cert with the relevant GlobalUserID (GUID), GlobalPrimaryGroupName (GGID) and GlobalSecondaryGroupNames (GUID) attributes depending on the users the installer requires. It is assumed that the GlobalVOID (GVID) is equivalent to the namespace set for the application across the data center. Figure 8 illustrates what a valid instance of an XOS-Cert should entail for the administrator in order to complete the installation process, using the Linux groups identified in the sample from sub-section 2.2.2. It shows that a system administrator issues a certificate to the application installer with the correct groups embedded. This is done else system administrators would have to add groups to their own identity certificate for every application to be installed. In addition, there is often a particular contact person that deals with customers on behalf of the data center, such that they (or more accurately their machine) assigns administrators to the management of different applications.

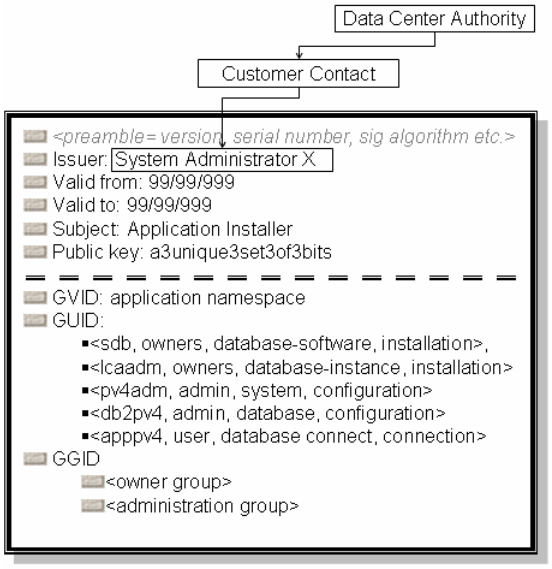

Figure 8: A hypothetical XOS-cert for the application installation example

The certification process would provide an answer for questions 1 and 2. However, what about questions 3 and 4? When are the groups created on the target nodes, how are the privileges associated and when are the user accounts (i.e. mapped from the GUIDs) added as valid users? These would have to be done by the end of Figure 7, step 8, as a resolution to the negotiation process between the originating XOS client and the target node. This suggests that the end of the negotiation process would mean sending the certificate of the application installer over to all target nodes, such that they can authenticate binaries belonging to the installer (as it is assumed any binaries are signed with the private key generated for the installer at step 1), as well as inspect the certificate for the groups, privileges and users to be created at the node. Depending on which aspect of the installation a node has been selected, the respective groups and users need to be installed. At this time, there is a need for each selected node to perform the mapping from strings in the certificate to POSIX GIDs and UIDs. The following Linux commands would then follow per node and per relevant group (some steps are advisories):

- `mkdir namespace`: create a unique directory for the application
- `chroot namespace`: set the root to that of the new directory created for the application
- `groupadd -g gid grpname`: create a group

- `useradd -d namespace -g gid -u uid`: create a user in a previously created group

However, what if these particular groups and users have already been mapped at the node and are persistent? Secondly, is it possible that a user or group be mapped to an already existing numeric identifier? Thirdly, especially for the case of the administrator privileges, what happens if a configuration setting required by the scheduled installation conflicts with that of another (e.g. set virtual memory size)[10]? Fourthly, as other customers are running, who may even have the same types of applications, the namespace might not be unique (such that the `mkdir` command would fail) and the machine cannot be re-booted or newly logged-into in order to effect changes or allow permissions to be active. These particular problems are beyond the scope of XtreemOS security currently, but could be addressed by including uniqueness checks for users and groups, as well as appending newly generated sequence numbers to the namespace. A more advanced, comprehensive solution would be to build the XtreemOS software bundle on top of a Linux distribution that contains support for secure isolation. One technique is that of virtualization [11], which would allow a clean, strong separation between different application namespaces and software running from different domains.

5.2.4 Summary and Outlook

This section discussed the deployment of legacy applications on top of an XtreemOS platform with the currently-specified security specification in place. Two areas that were more carefully scrutinized are:

- ensuring that identity management and deployment mechanisms do not break legacy application installation
- checking that the current user concept remains compatible with POSIX standards

In order to simplify the deployment of legacy applications, an administrator could make the decision to introduce single superusers on each node known as *pilot jobs*. Any application installer would just have to be handed this user id and run in its context. This would get rid of some of the above discussed technicalities but would create a large vulnerability as the separation of privilege, least privileges and least common mechanism principles would be broken. The above process is therefore recommended, including the possible extensions to include virtualization.

[10] One workaround here would be to use a maximum policy, where the virtual memory is set to support the demands of the highest requester

Other Alternative Actions: Since there is (i) no backward propagation of local user/group operations and (ii) no privileged operations at all for legacy installers, a conflict with the current design could only be avoided by using a segmentation of the system user and group id ranges. One range could serve legacy applications, the other range could serve Grid jobs. A privileged pilot job could be used to deploy the application via their installers. However this does not solve the security problem. An attribute in a Grid user identity certificate could be used to indicate if the user needs a static mapping for legacy applications or the current proposed mapping. However this attribute should belong to the application and not to the user since a user can run different kinds of applications.

Still this is a workaround which may be enough for executing parts of legacy applications to exemplify other XtreemOS system attributes. It is not a comprehensive solution to this problem and requires a high and error prone manual effort. Avoiding the superuser idea

Why Isolation is relevant but complex Deploying applications, selecting appropriate resources to support them and managing the runtime environment is a complex problem[8]. Adding the handling of confidential data to the equation makes it even more complex. The topic of isolation will receive more attention as the specification matures, but has already been mentioned frequently. The interesting challenge for XtreemOS here is that now jobs, applications and processes are being executed in the context of an operating system user as opposed to in the same context and uid of a single grid scheduler. Isolation requirements arise as multiple schedulers, processes, applications and jobs, from different sources and with different users may be running on the same nodes. We therefore understand that a closer look at isolation mechanisms is required for the following reasons:

- Size of the code base: The Linux code base with the supporting libraries are huge. From the beginning up to the mid of year 2007 about 125 security vulnerabilities are reported at the kernel level (see www.securityfocus.org).
- There is no separation of name spaces on the level of user context and no access control mechanism to allow a fine grained setting apart from read/write/execute.
- System characteristics have to be segregated and users need the illusion of being alone on that platform. With legacy and business applications there is a need to change system variables, which might affect the execution of other applications being run in parallel.

Since the deployment in XtreemOS via AEM is the single point of entry into XtreemOS nodes, there are some potential problems for isolation that may arise

here. One step in this direction could be the integration of SELinux[12] extensions into the Linux distribution in order to provide mandatory access control capabilities.

6 Conclusions

This specification presents a design for managing global entities in a XtreemOS VO environment, and for the first time, a complete security architecture to enable secure application execution and file management in XtreemOS. We have also described some insights into the future direction of our security work and identified a number of potential stumbling blocks for deploying and managing complex applications in XtreemOS. This process has been challenging and fruitful.

However, it should be noted that the work presented in this specification is based upon our current understanding of the design and implementation of the system components provided by other WPs. It also reflects the ongoing and still highly active cross-WP interactions that have been taking place in the project to ensure our work satisfies the security challenges presented by other WPs. We will continue to work closely with our colleagues in WP2.1 to ensure the Grid level security services that we have been developing will be adequately supported by the node level operating system mechanisms to realise a scalable and secure approach for providing native VO support in XtreemOS.

References

[1] XtreemOS Consortium. Requirement documentation and architecture for xtreemfs. In Felix Hupfeld, editor, *XtreemOS public deliverables - D3.4.1*. Work Package 3.4, November 2006.

[2] XtreemOS Consortium. Design and implementation of node-level vo support. In *XtreemOS public deliverables - D2.1.2*. Work Package 2.1, November 2007.

[3] XtreemOS Consortium. Design of the architecture for application execution management in xtreemos. In *XtreemOS public deliverables - D3.3.2*. Work Package 3.3, May 2007.

[4] XtreemOS Consortium. *Evaluation Report and Revision of Application Requirements*. Work Package 3.3, December 2007.

[5] XtreemOS Consortium. First specification of security services. In *XtreemOS public deliverables - D3.5.3*. Work Package 3.5, May 2007.

[6] XtreemOS Consortium. *The First Version of System Architecture*. Work Package 3.1, December 2007.

[7] XtreemOS Consortium. Xtreemfs prototype month 18. In Toni Cortes, editor, *XtreemOS public deliverables - D3.4.2*, 2007.

[8] Harry J. Foxwell and Isaac Rozenfeld, 2005.

[9] Matthias Hess. Xtreemfs and security - slides presented in the wp3.5 abingdon meeting. XtreemOS Internal Communication, Oct. 2007.

[10] R. Housley, W. Polk, W. Ford, and D. Solo. Rfc 3280 - internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile, April 2002.

[11] Hans Löhr, HariGovind V. Ramasamy, Ahmad-Reza Sadeghi, Stefan Schulz, Matthias Schunter, and Christian Stüble. Enhancing grid security using trusted virtualization. In Bin Xiao, Laurence Tianruo Yang, Jianhua Ma, Christian Müller-Schloer, and Yu Hua, editors, *ATC*, volume 4610 of *Lecture Notes in Computer Science*, pages 372–384. Springer, 2007.

[12] Stephen Smalley, Chris Vance, and Wayne Salamon. Implementing SELinux as a Linux security module. NAI Labs Report #01-043, NAI Labs, Dec 2001. Revised May 2002.

[13] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Rfc 3820 - internet x.509 public key infrastructure (pki) proxy certificate profile, June 2004.