Project no. IST-033576

XtreemOS

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

Design and Implementation of Node-level VO Support
D2.1.2

Due date of deliverable: November 30th, 2007
Actual submission date: December 18th,2007

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP2.1
Task number: T2.1.3

Responsible institution: ICT
Editor & and editor's address: Haiyan Yu
Institute of Computing Technology
No.6 Ke Xue Yuan Nan Lu
100080 Beijing
China

Version 0.11 / Last edited by ICT Team / Nov 28th, 2007

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
0.1	10/03/07	ICT team	ICT	Initial draft
0.2	20/03/07	ICT team	ICT	Architecture
0.3	26/03/07	ICT team	ICT	Scenarios, overall modification
0.4	03/04/07	TID Team	TID	Section on NSSWITCH, some comments added
0.5	10/04/07	An Qin	ICT	Architecture and Implementation revising, added PAM module architecture, authentication mechanism, and some comments
0.6	01/11/07	An Qin	ICT	Rewrite chapters of PAM, Interface, Implementation, add chapters of AMS and Applications
0.7	05/11/07	Haiyan Yu	ICT	Major update of main sections for improved organization of content
0.8	09/11/07	Haiyan Yu	ICT	Overall modification
0.9	12/11/07	Luis Pablo Prieto	TID	Proofreading and small modifications throughout the document
0.10	20/11/07	Luis Pablo Prieto	TID	Fixed incoherent part in NSS section
0.11	20/11/07	Yvon Jégou	INRIA	Proofreading and small typos
0.12	28/11/07	Haiyan Yu	ICT	Incorporate internal reviewers' comments

Reviewers:
Guillaume Pierre (VUA), Philip Robinson(SAP)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.1.3	Design and implementation of basic version of node-level VO support mechanisms	ICT*, INRIA, TID, STFC, CNR, SAP

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

Contents

Executive Summary

This document describes the design and implementation of node-level VO support mechanisms in Linux-XOS, an XtreemOS flavor for stand-alone PC nodes. The architecture design is presented firstly to give an overview of both internal working of components for node-level VO support and their interactions with high-level XtreemOS services. Further detailed design and implementation strategies for each component are discussed subsequently. Lastly, external interfaces to exploit VO support functionalities are given according to different targeted user roles, including *normal VO users*, *administrators* and *developers*.

The main objective of node-level VO support is to provide necessary local user and resource management extensions to enable secure and effective sharing of a PC node among multiple VOs. The key issues to be addressed are: 1) Linux system should be able to recognize and process VO user identities and additional attributes. 2) Multiple VO users' access of a node must be differentiated and isolated. 3) VO users' consumption of resources must obey VO-level and node-level policies of access control and usage constraints. Following the design principles presented in D2.1.1 [7], several standard Linux system services and utilities are chosen as extension points for embedding VO support functionalities, avoiding the patching of kernel code, while providing backward application compatibility.

In current Linux there are two important mechanisms related to user management: Pluggable Authentication Module (PAM) and Name Service Switch (NSS). PAM and NSS are both extensible frameworks that allow new user authentication methods or name resolving schemes to be plugged into Linux easily. VO support functionalities are implemented as specific PAM and NSS extensions together with an auxiliary runtime service, the Account Mapping Service (AMS). These extensions enable applications to process VO-level user information via standard PAM and NSS APIs. VO users are dynamically mapped into local user accounts during PAM conversations, and the mapping information can be fetched via NSS APIs. AMS is designed to serve as the back-end for PAM and NSS extensions, which does the actual mapping operations based on local configured mapping rules. AMS also acts as the local policy engine for enforcing both VO-level and node-level policies.

As a demonstration and proof-of-concept of our approach, OpenSSH, the most widely used shell tool for Linux users, is modified to use newly designed PAM/NSS extensions to allow VO users interactively access a remote XtreemOS node.

1 Introduction

In XtreemOS, a VO is a dynamic coupling of multiple Linux nodes for resource sharing under specific polices. The policies specified by a VO, such as security, resource usage limitations and scheduling priorities for shared resources, will be finally checked and ensured at individual nodes by the local Operating System (OS) instance. The main objective of node-level VO support is to provide necessary local user and resource management extensions that facilitate the secure and efficient sharing of a local node among multiple (possibly dynamic) VOs. Key issues to be addressed are:

- The Linux system should be able to recognize and process VO user identities and additional attributes (e.g. VO user roles). The approach of mapping from VO user identities to local Linux user identities needs to be scalable enough to support the dynamic changing of VOs.
- Multiple VO users' access of a node must be differentiated and isolated. It is necessary to keep OS objects (e.g. processes, files, IPC resources) owned by one VO user from interfering with others.
- VO users' consumption of resources must obey VO-level and node-level policies of access control and usage constraints. A node can provide resources to multiple VOs at the same time but with different constraints.

Our design principles have already been presented in D2.1.1: i) Changes to existing Linux codes should be minimized to provide backward compatibility, especially changes of kernel code must be avoided in the first version, and ii) The architecture should be designed to be flexible, not only making it possible to adapt to a diversity of existing or new VO models, but also facilitating code maintenance.

This document is organized as follows: Section 2 presents the overall architecture design. Sections 3, 4 and 5 discuss the design and implementation strategies of each component in more detail: the PAM module, the NSS module and AMS, respectively. Finally, Section 6 details the external user interfaces and section 7 concludes with a roadmap of future works.

2 Overall Design

2.1 Native VO Support – Where to Start ?

As changes to the kernel were ruled out from the beginning, node-level VO support has to provide mapping from VO-level identities and policies, to their local counterparts that can be fully recognized and processed by Linux. While this mapping could be simply done by a shell script that designates a local user name (or call `useradd` to create new one) against a VO user name like a Distinguished Name (DN), the selected solution must meet the following requirements:

Security The mapping should be done in a secure way. The caller of the mapping process needs to be a trusted party. VO user's credentials must be validated to ensure that the claimed VO user name and attributes are trustworthy. The mapped local accounts must be bound with kernel-acknowledged security attributes that conform to VO access control policies. The runtime mapping information is to be securely maintained to resist various attacks. Processes launched by the mapped account must be attached with its corresponding VO user's identity information, for the sake of non-repudiate tracking and logging resource usage by the VO user.

Scalability The mapping mechanism should be scalable to support dynamic changing inside a VO. For example, if participants of a VO or policies of the VO change during runtime, how does the mapping data (e.g. allocated local accounts and associated local security configurations) in each participating node of the VO reflect the change? It would be a nightmare for each node administrator to manually maintain local system to ensure this kind of consistency.

Flexibility How does the mapping support different types of VO user credentials? Currently XtreemOS adopts a proxy certificate (i.e. XOS-Cert [8]) as the default user credential, but it would be helpful to introduce a flexible framework that could adapt to other VO models. This could facilitate the interoperability between XtreemOS and other Grid systems. Another question is how are different mapping rules applied? For example, a VO user could be mapped to an existing local account, or a random allocated account together with a virtual machine created on the fly. A unified flexible framework is necessary in such case to make the adopted mapping scheme independent of applications.

Compatibility How do applications exploit this mapping functionality? Do they need to change their codes to fit with new APIs? For a legacy Linux application (e.g. `Apache`, or `proftp`) to provide sharing service for a VO, it is meaningful to let the application directly authenticate with VO users and authorize based on VO policies. In such case it may not be a reasonable choice to force them to modify their implementations to be *VO-aware*.

To answer the above questions in mind, we have identified several important existing Linux mechanisms for achieving VO support from the Operating System (OS) perspective. Pluggable Authentication Module (PAM) [12], Name Service Switch (NSS) [6] and Kernel Key Retention Service (KKRS) [3] are important mechanisms tightly related to user and process management in current Linux. PAM has become a de-facto standard for authenticating users, which could be considered as the first entry point of user's access to the local system. User-related information is processed by `libc` APIs that could be intercepted by NSS modules. Custom PAM and NSS modules can be deployed by system administrators without changing the kernel, and remain compatible with legacy applications. KKRS provides a facility to store security data in a process's control block in kernel and keep tracking of it across the process's descendants. These mechanisms provide the required features for developing new VO support extensions.

2.2 User Mapping from VOs to Local OS

Depending on the VO model, the credentials of VO users might be different (e.g uid/pwd, long-term/short-term certificates, Shibboleth handles [9], or pass token like Kerberos tickets [1]). And also how VO users will be mapped onto local accounts varies in many ways (e.g. mapping to temporarily allocated accounts, to an account pool, or to accounts even bundled with virtual machines). By implementing node-level VO support functionalities as specific PAM/NSS modules, the local system could accommodate different VO models in a '*hot-plugged*' manner. For example, for dealing with each kind of credential format or mapping scheme, proper PAM/NSS modules could be choosen by the system administrator.

Scalable management of dynamic changing VOs is made possible in XtreemOS by dynamic allocation of local UID/GID(s) on nodes, according to user name and attribute information stored in VO users' XOS-Certs. The number of local accounts needed is bounded by the number of users simultaneously accessing the node, regardless of the overall number of VOs and of users per VO. The allocated local accounts and their bundled local policies will be revoked once the user's access terminates, either marked by user's logging out or termination signals explicitly sent by job management service. The runtime updating of mappings and policies could be possible provided that there is a back-end service tracking all mapping relations and associated user processes.

2.3 User Authentication

PAM integrates multiple low-level authentication technologies (USB dongles, Kerberos, SQL-based authentication...) into a common high-level API. Applications requiring authentication can be developed independently of the underlying authentication mechanism. PAM technology, originally proposed by Sun, is now widely used in Linux distributions, and it is exploited by XtreemOS to interface with the Linux and Grid authentication services.

XtreemOS-specific PAM modules (`pam_xos` for short) can be developed to apply sophisticated policies in three phases of service execution: authentication, authorization and session management. With `pam_xos`, VO users are authenticated using their XOS-Cert credentials, and VO policies are enforced during the authorization phase. Session management in `pam_xos` implements dynamic management of local accounts, with automatic housekeeping of local files and processes, credential management and name service updates. It is noteworthy that a PAM module is actually a shared library called by applications, i.e. it coexists with applications and does not keep persistent records in a separate space. Therefore, a special auxiliary service, the Account Mapping Service (AMS), was designed to serve as the back-end of PAM for runtime management of mappings.

2.4 Naming Conversion

NSS is a mechanism for intercepting queries to traditional Unix file-based information databases (e.g. `/etc/passwd`, `/etc/group`), substituting them with other databases, like NIS+, LDAP or (in our case) custom information systems. NSSwitch has been included in the GNU C library (`libc`) used by Linux.

In XtreemOS, the NSSwitch module translates account-related information according to the PAM-established mapping. The information is maintained in databases accessible through the local AMS. By embedding account resolution into an NSSwitch module, we remain compatible with legacy applications, and we restrict access to grid user's account information to authorized users within the VO.

2.5 Access Control and Logging

Mapping VO users to local accounts allows a degree of isolation among VO users. Undoubtly the activities of mapped local accounts must be stamped with their corresponding VO users' credentials. The KKRS allows caching of authentication data related to a process within the kernel. Other kernel services, including file systems, can access this information and delegate operations to authenticated user-space applications.

During session initialization, the PAM stores the user's XOS-Cert in the kernel session keyring: this will be associated to all local processes generated from the user request, and will be retrieved each time the VO user credentials need to be used. PAM-aware OS services, and XtreemOS ones in particular, can transparently check VO user authorization, using the credentials from the session keyring. Local service auditing and resource usage accounting are also possible. The audit log must contain references to user credentials and be securely provided to the resource owner as well as the VO manager.

When the XtreemOS PAM module does a mapping between VO-level credentials and node-level UID/GIDs, it can add specific information in the kernel keyrings. Fine grained access control and auditing on operating system objects (e.g. processes, sockets) requires the support of the kernel, and can be provided, for example, through LSM (Linux Security Modules). Fine grained access control is possible even when the application activity generates requests to external services. This is the case for XtreemFS or NFSv4 filesystems, as `pam_xos` can store secondary group lists, tickets, and file-related credentials in the user's kernel keyring.

2.6 An Architectural View

Figure 1 shows an architecture overview of node-level VO support. A VO user would obtain an XOS-Cert from a VO manager (e.g. currently from the Credential Distribution Authority[8]) and present it to the user application on a PAM-aware resource node in the VO. XOS-Cert would be checked for validity by the PAM, and stored in the KKRS, associated with the user process. Thus, this process and all its children can show that certificate to local and remote services.

Once authentication is over, whenever a process requires user or group information (using standard `libc` calls), it will go through the NSS subsystem. The NSS module will obtain the authorization information from the KKRS, and ask the AMS for the requested information. The AMS will grant that information depending on the authorization data, which is then returned to the process.

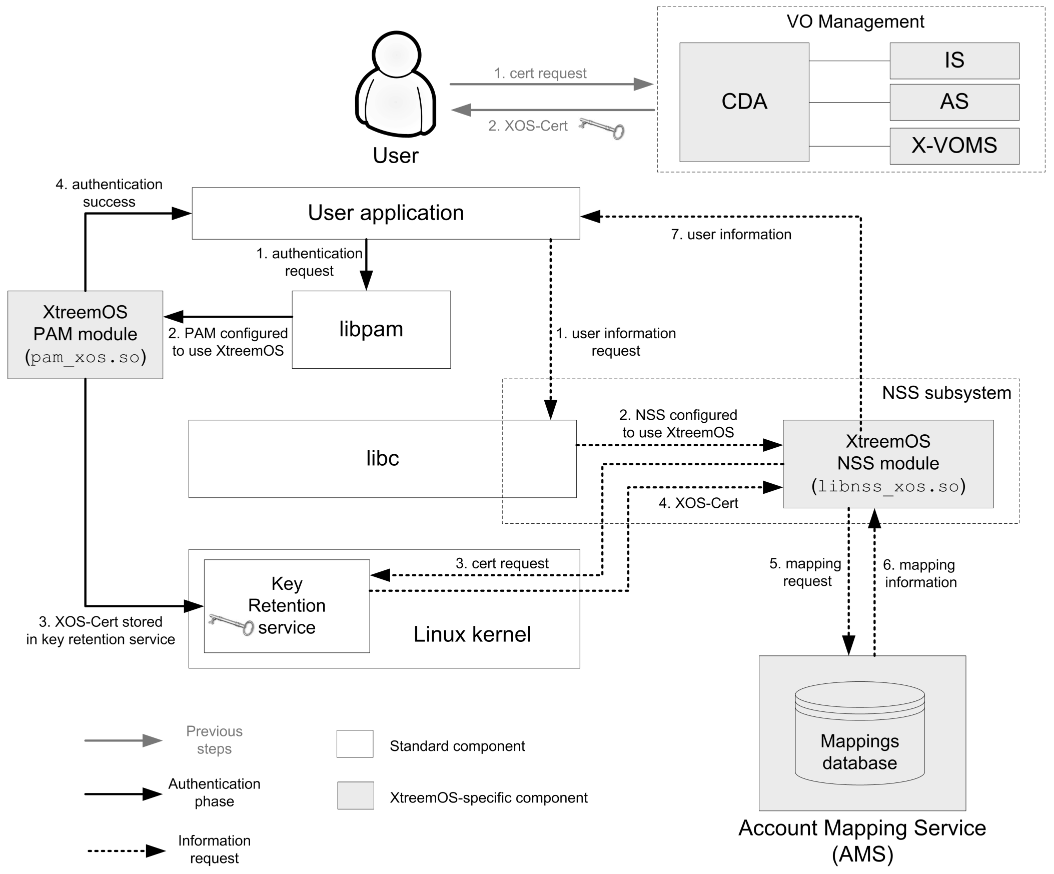
VO Management
CDA
IS
AS
X-VOMS
User
1. cert request
2. XOS-Cert
4. authentication success
User application
1. authentication request
XtreemOS PAM module (pam_xos.so)
2. PAM configured to use XtreemOS
libpam
7. user information
1. user information request
NSS subsystem
libc
2. NSS configured to use XtreemOS
XtreemOS NSS module (libnss_xos.so)
4. XOS-Cert
3. cert request
Key Retention service
Linux kernel
3. XOS-Cert stored in key retention service
5. mapping request
6. mapping information
Mappings database
Account Mapping Service (AMS)
Previous steps
Authentication phase
Information request
Standard component
XtreemOS-specific component

Figure 1: An Architectural View of Node-level VO support

3 XtreemOS PAM Extension

3.1 Overview of the PAM approach

PAM is designed to allow system developers to add customized authentication mechanisms into applications without changing application source codes, provided that these applications are using PAM APIs for authentication, i.e., they are PAM-aware. PAM is already supported in almost every Linux distribution (it is a de-facto standard).

The core of PAM consists of a set of abstraction interfaces that cover full stages (also called services in PAM) of a security checking procedure: *authentication*, *account management*, *password management* and *session management*. These interfaces are called by PAM-aware applications, while the actual implementation of interfaces is done by PAM modules. For each application, the set of PAM modules to invoke is configured externally in a file, generally within `/etc/pam.d/`. Multiple PAM modules can be chained together for a single purpose or service.

3.2 Developing the XtreemOS PAM Module

The task of dealing with VO user identities in a local node is mainly done by a XtreemOS PAM module, namely `pam_xos.so`. PAM interfaces called by applications (accordingly, implemented by PAM modules) are described as follows :

- `pam_authenticate` authenticates a user. Passing application-defined data between a PAM module and the application could be done via the `pam_conv` struct. In the XtreemOS case, user certificate data is passed from applications to `pam_xos.so` by this way.
- `pam_setcred` alters the credentials of the user, after the user has been authenticated but before a session has been established. Additional information about the user can be provided here to a PAM-aware application. In the XtreemOS case, VO user identity related information is stored into keyring during this phase.
- `pam_acct_mgmt` performs the task of determining whether the user is permitted to gain access after the user has previously been authenticated. It is another place to further check the validity of users with respect to other information (e.g. whether the claimed account exists or whether the user is authroized to switch to that account). This interface is important for XtreemOS as the account mapping is done here.

- `pam_open_session` and `pam_close_session` are doing initialization and cleanup works for a session, respectively. A session is a period of activities (e.g. from user logged in to logged out) after the user has been authenticated and granted access. These two interfaces are useful to do session-wide policy setup/cleanup and allocation/release of temporary resources.

- `pam_chauthtok` is used to (re-)set the authentication token of the user (e.g. changing password).

Almost every PAM interface mentioned above is exploited by `pam_xos.so` to insert VO support functionality. Any PAM-aware application is able to verify and process VO user information and attributes, provided that it is configured to use `pam_xos.so.`

3.3 How It Works

Fig. 2 illustrates how the PAM module works with internal components as well as high-level XtreemOS services.

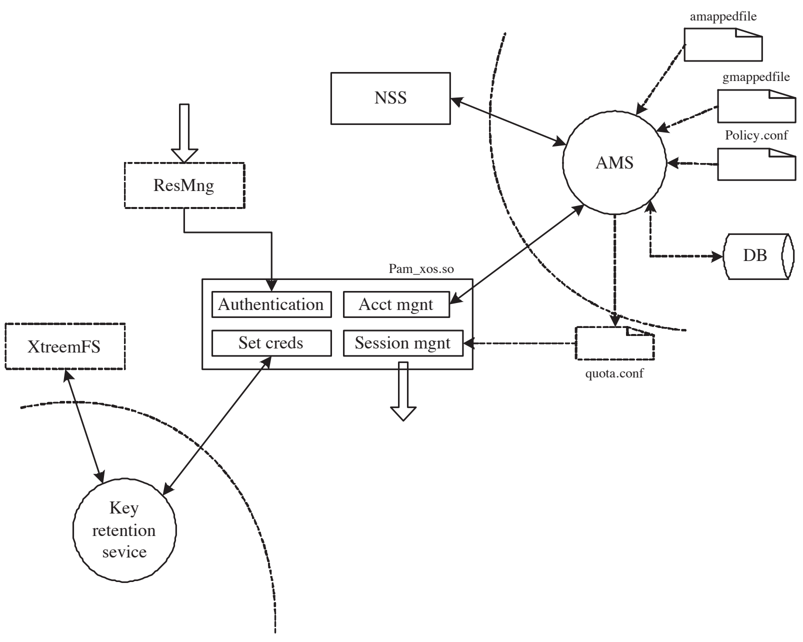

Figure 2: The XtreemOS PAM module

User Authentication The main loop of a PAM-aware application starts with the `pam_authenticate` interface provided by the PAM module, which is designed to verify whether (proxy) certificates of VO users are issued by a trusted CDA and check whether users are allowed to access the local node according to the configured VO policy.

- Certificate verification following the standard of RFC3820 [10] is performed by checking: i) basic certificate information (e.g. signature, validity period, issuer's name and subject name), ii) certificate extension (if the ProxyCertInfo extension exists), and iii) certificate chain. The root of the chain must be the certificate of a trusted CDA.
- After the verification of user certificate, the PAM module will look up access control policies specified in a local file `policy.conf`. Currently, a very coarse level of access control is supported. The policy specifies whether a VO user (or a VO group, or a given role) is allowed to access the node.

If any of the above checking fails, the authentication process will terminate immediately with a `PAM_AUTH_ERR` value returned to the application.

User Mapping To manage the runtime user mappings in a de-coupled way, the Account Mapping Service (AMS) has been designed. AMS is a daemon service which performs the actual user mapping actions and serves as the back-end for the PAM module and the NSS module. After the authentication step, VO user identity information together with attribute information (e.g. *group* or *role* of the VO user) could be fetched from user certificate and the control logic goes to another PAM interface `pam_acct_mgmt`. The PAM module then contacts AMS via a socket connection by sending all fetched user information. AMS does the mapping by creating local user accounts (and possibly groups) and responds to the PAM module with a `passwd` struct that stores the information of mapped accounts (e.g. uid/gid(s)). AMS also sets up the local policies when creating local accounts. Hereafter the PAM module could do some enforcement works in its logic (e.g. in `pam_open_session`).

Three configuration files are used by AMS: `amappedfile` (for account mapping rules), `gmappedfile` (for group mapping rules) and `policy.conf` (for local access control and resource usage policies). Currently mapping to a dynamically created account (the default rule) or a pre-allocated account is supported. Preliminary support of resource usage policy enforcement is considered which relies on the system call `setrlimit()` [1].

[1]More fine-grained policy enforcement mechanisms are discussed in Section 5.3

AMS maintains runtime data of user mappings which could be queried by NSS APIs. For the sake of security, the direct manipulation of data maintained in AMS is not possible. The only service trusted by AMS is the PAM module (`pam_xos.so`), which is able to modify mapping information. NSS and other high-level PAM-aware applications can only fetch data from AMS (i.e. read-only access).

Interactions with high-level XtreemOS services The PAM module stores user credentials (i.e. XOS-Cert) into kernel key retention service (KKRS), which could then be accessed by XtreemFS for performing access control on global files. According to the overall architecture of XtreemOS, VO users get their credentials from security services and pass them to the job management services, namely the Application Execution Management (AEM). There is a component in AEM, the Resource Management Service (ResMng in Fig 2), deployed in each node for negotiating with the local OS. To process VO user information, ResMng is to be implemented as a PAM-aware application.

Fig. 3 depicts a sequence chart of the collaboration between ResMng and the PAM module. Firstly ResMng relays user credential to the PAM module. Then the PAM module checks the validity of user credential in the *authentication* part. If the verification process is successfully done, the control goes to *account management* part, which contacts AMS for requesting a `passwd` struct (i.e. to hold the information about local mapped user). AMS processes the request by performing the task of user/group mapping and setup local policies associated with the mapped users/groups. After the mapping has been accomplished, `pam_setcred` puts all related identity information (including XOS-Cred and mapped uid/gid) into keyring. Lastly ResMng is able to call `pam_open_session` function to start the procedure of granting resource access. The mapped uid/gid returned to ResMng could be used for launching jobs with local user identity (e.g. by `a setuid` program or `sudoers`).

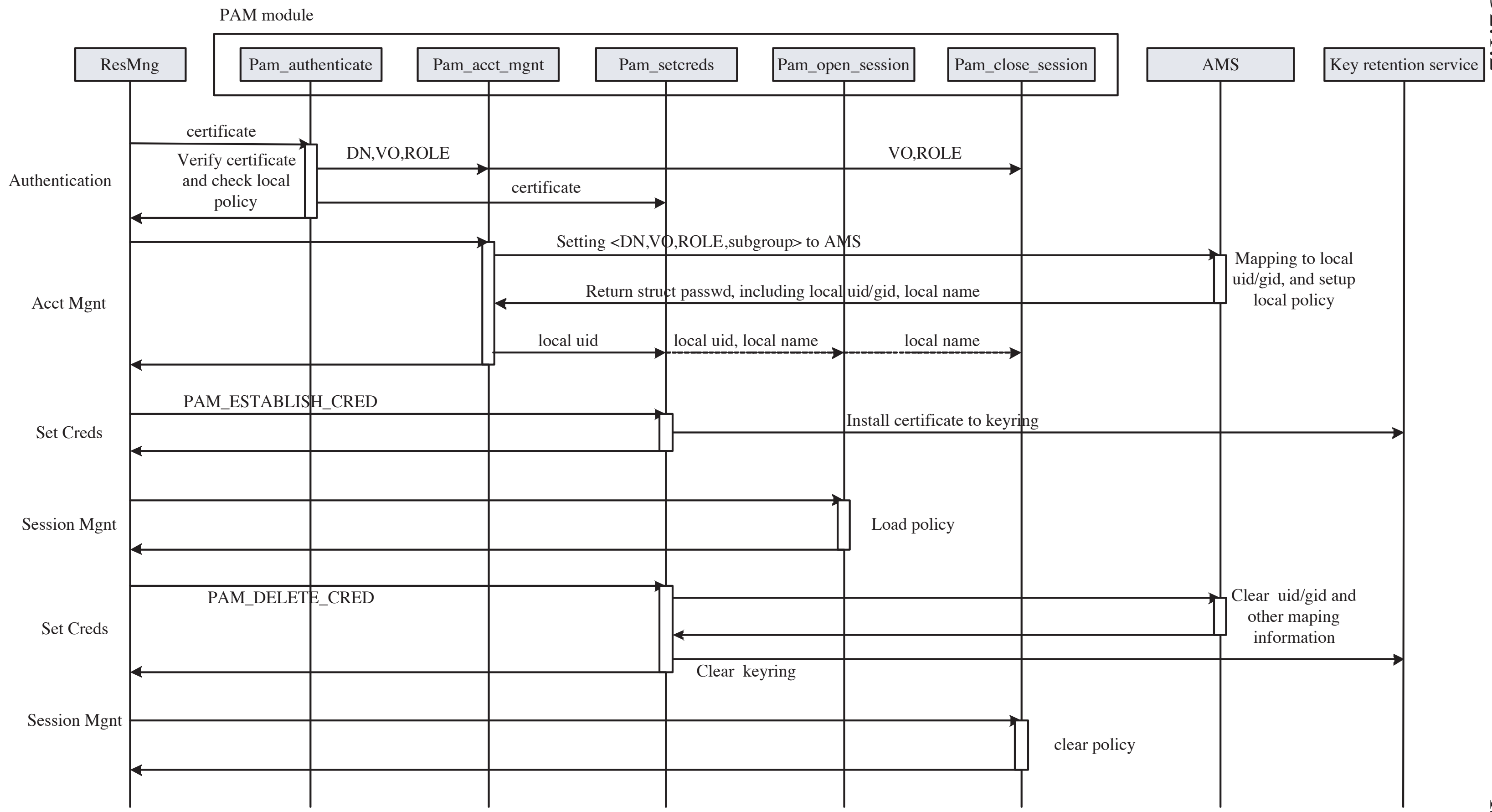

Figure 3: Integration of ResMng and the PAM module

4 XtreemOS NSS Extension

4.1 Overview of the NSS approach

The Name Service Switch (NSS or NSSWITCH) was designed so that the behavior of several libc functions that need to retrieve information from databases could be dynamically controlled and personalized without having to modify any application's source code.

When an application needs to obtain information about a user (for instance its user name, home directory, default shell, etc.), the standard way to obtain this information is calling the function `getpwnam(username)`. This function is *not* a system call but a libc function.

Traditionally, what `getpwnam()` does is open the file `/etc/passwd` and look for the user's `username` in this file. With NSS, this behavior can be changed and this information can be retrieved from many other sources (such as the Network Information Service (NIS), an LDAP directory, a SQL database, and so on).

The following are libc functions whose behavior can be controlled via NSS and that will be of interest for XtreemOS:

```
struct passwd *getpwnam(const char *name)
struct passwd *getpwuid(uid_t uid)
struct passwd *getpwent(void)
struct group *getgrnam(const char *name)
struct group *getgrgid(gid_t gid)
struct group *getgrent(void)
```

These functions return information about users and groups. Users and groups can be searched for according to their name (`getpwnam()` and `getgrnam()`), their UID/GID (`getpwuid()` and `getgrgid`), or they can just be fetched from the database one by one (`getpwent()` and `getgrent()`).

4.2 Developing the XtreemOS NSS Module

XtreemOS can use NSS to provide applications with coherent information related to VO users. Whenever applications request information about VO user names or groups, the request is handled by XtreemOS services and meaningful information can be given to the user provided that he/she holds the right permissions. In this way, legacy XtreemOS-unaware applications can be made VO-aware to the maximum possible extent without the need to modify their source code. By using NSS, system commands such as '`ls -l`' or '`who`' will show meaningful information (e.g. the VO user ID instead of the numeric UID) without having to be recompiled. This is essential to give the user the impression of being using a Grid-enabled operating system.

The behavior of NSS will be dependant on the VO-model and policy. Some VO models will require NSS to return the user's Distinguished Name(DN) as username (or more user-friendly an email address), whereas other may require NSS to return a meaningless user name in order to hide the user's identity.

XtreemOS-aware applications will be able to access more advanced user information (such as the current user's location, running jobs, etc.) via an extended API such as SAGA.

4.3 How It Works

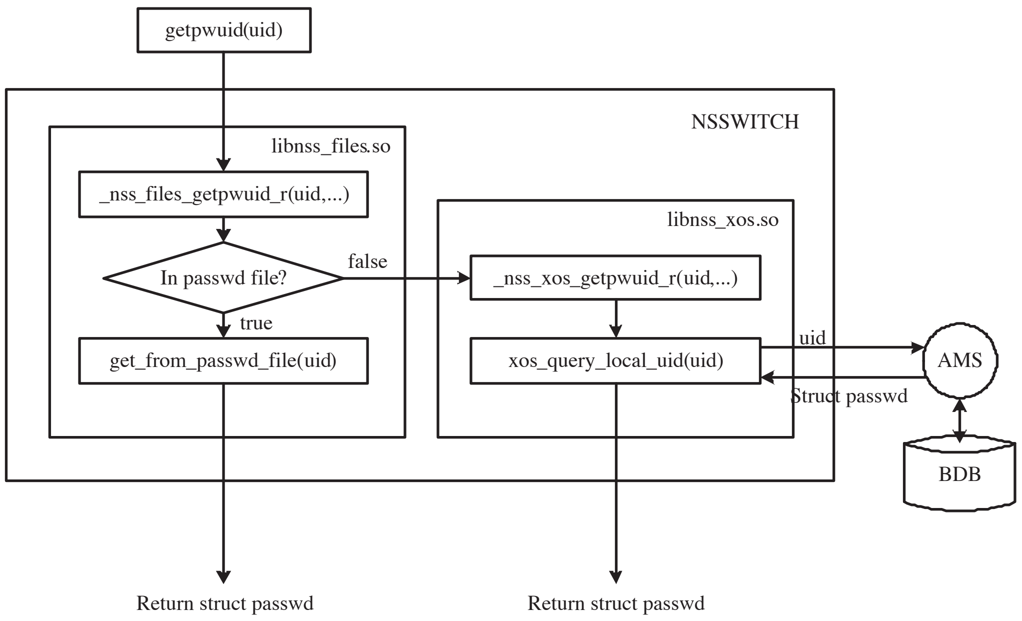

Figure 4: Proposed Flowchart of the NSS Module

Due to the concept of global (VO) users, a NSS module is needed by XtreemOS. It will be necessary to implement (at least) the `password` and `group` databases so that user name and group information about VO users can be retrieved from applications, by using functions such as `getpwuid` or `getgrgid`. The XtreemOS NSS module will need to communicate with the local mapper service (the AMS), a local daemon that will be in charge of storing and managing the mapping table with all the VO users currently in the system and the UID/GIDs that they are locally mapped to.

The NSS module must make use of XtreemOS services with the user's credentials (which will be stored in his/her keyrings) and be aware of XtreemOS security mechanisms.

The flowchart in figure 4 shows the inner workings of NSS. When an application running on behalf of a global (VO) user queries about a user or group information using its name (via `getpwnam()` or `getgrnam()`) or its UID/GID (via `getpwuid()` or `getgrgid()`), the call will be handled by one or more NSS modules, as configured by the local administrator in `/etc/nsswitch.conf`:

```
$ vi /etc/nsswitch.conf
...
passwd:         compat xos
group:          compat xos
...
```

When NSS is configured as shown above, the call is first handled by the usual `compat` NSS module, which looks for the information in the local `/etc/passwd` and `/etc/group files`. If the user is found in these files, the information in those files is returned, making legacy applications to work exactly as on an unmodified Linux. But, if the user is not found in the local files (and thus, it is a VO user *not* mapped to an existing account), the call is forwarded to the xos NSS module, which will issue a request to the local system mapper (AMS), to obtain the necessary information about the global user or the virtual organization. This information will be mapped to traditional UNIX structures (the `struct passwd` or `struct group` structures) so that the applications can work in the same way as ever, without changes to their source code.

The configuration shown above is the recommended behaviour of the NSS subsystem in XtreemOS. It must be noted that the order of the modules (i.e. `compat` and `xos`) is not important, in the case of mapping a global (VO) user to a pre-existent local account. With the configuration above, the data returned by NSS will be the local data as stored in /etc/passwd, while changing the order of the modules will produce the VO information of the user in all mapping cases, *even if the global account is mapped to a local existing account*. However, there can be cases where this behaviour is desired by local administrators, and thus, it has been left open for them to decide (although the recommended mode is configured by default).

4.4 Open Questions

Complex format of usernames Some legacy applications may have issues with usernames containing spaces, commas, and other characters. To handle this, we can implement different mappings from DNs to conventional Linux usernames and let the VO-model or policy choose which mapping strategy best suits their needs.

`getpwent()` function The `getwpent()` function is supposed to return information about all the users in the system one-by-one. It is possible to enumerate all VO user information by contacting high-level VO management services, or at least, the current active VO users maintained in AMS. Moreover this kind of exhaustive query must be limited (e.g. limit this function to return users in the same VO and limit the requester to VO admin).

5 Account Mapping Service

5.1 Overview

The Account Mapping Service (AMS) plays the crucial roles of managing runtime user mappings and acting as local policy engine for VO access on the node. It is designed as a separate daemon service running with root privileges so as to decouple the core VO support functionalities from specific PAM/NSS modules. The benefits of introducing such a design are:

- The AMS service is a centralized point to ensure the consistency of local mapping information. As PAM and NSS modules are all shared libraries that processing data in memory space of their driving applications, it is necessary to have a back-end service to securely maintain key mapping data with persistence support (e.g. in a database).
- As AMS could be configured to start at system boot time, user mappings and local policies handled by AMS could be dynamically changed at runtime. This could facilitate the support for dynamically changing VOs in terms of adding/removing resources or adjusting scheduling policies. AMS itself could be refreshed with an updated configuration file (e.g. by sending `SIGHUP`) even when any depending applications have already started running.
- It allows a graceful integration of VO support functionalities into LinuxSSI. Only one unique mapping service is needed for nodes of the whole SSI system, though PAM/NSS modules could be required to be deployed on each node.

As a summary of aforementioned functionalities, AMS is responsible for the following tasks:

- map VO users to local ones. According to mapping rules defined by local system administrator, AMS determines how to allocate local *uid*, *gid(s)* and possibly *home directory* for a VO user.
- enforce local policies on resource access control and usage constraints.
- maintain the mapping information and local policies to ensure that they are securely stored and consistent with VO users' current status.

5.2 User Mapping Management

By default, a VO user is mapped to a dynamically created local account (and associated local groups if needed) when a session starts. When the VO user terminates the session (or the session is closed by a PAM-aware application), the temporary account is deleted, and local configuration data associated with this account are cleared. VO users could also be mapped to a pre-existing account.

The current syntax of a rule to specify how VO users are mapped onto local users is as follows:

```
<vo-subject> <local-subject> <init-driver>
```

where `vo-subject` is an expression (e.g. *regular expression* or simpler *glob* style string) to match VO identities (`VO id`, `user id`, `groups` and `roles` carried in XOS-Cert), `local-subject` is the local account or group onto which `vo-subject` is to be mapped, `init-driver` is an optional callout (e.g. a script) that helps to set up the local mapped account/group. The `local-subject` could be a wildcard character * which means that a random account/group will be created for the mapping, or it could be an existing local account/group name. If `init-driver` is specified, the local mapped account/group information will be passed into it to perform additional actions (e.g. allocate a virtual machine slice of the node to the mapped user).

5.3 Policy Management and Enforcement

AMS is an appropriate place to perform local policy management and enforcement. To facilitate the secure and efficient sharing of nodes among VO users, it is necessary to provide built-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 natural to use Access Control Lists (ACLs) to specify more fine-grained file access policies. Sockets can 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 first. During the mapping process, access control policies are populated into local security configurations such as file permission bits and ACLs, which are 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 [13] 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 [5], KVM [2], OpenVZ [4], Xen [11] are available for use, which would provide extensive support for system resource partition. Currently which approach will be leveraged by XtreemOS has not been decided yet.

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.

Application-specific policies It is necessary to allow local admins to express application-specific policies about 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 can only be recognized by `app-policy-driver`. When the callout is launched, the mapped local account information corresponding to `subject` will be passed into it.

6 Interfaces

This section describes external user interfaces to exploit VO support functionalities developed in WP2.1, in terms of command line tools, configuration files and application programming interfaces (APIs). These interfaces are classified according to targeted user roles: *VO users*, *node administrators* and *developers*.

6.1 Interfaces for VO Users

Generally speaking, low-level VO support functionalities are hidden from VO users when accessing grid resources. Normally VO users should only contact high-level services like Application Execution Management (AEM), which indirectly incorporates node-level VO support mechanisms.

One exception is the shell tool OpenSSH. OpenSSH is the most widely used secure shell tool to remotely access Linux/Unix hosts and a straightforward data moving tool (e.g. `scp/sftp`) among hosts. OpenSSH encrypts all traffic and provides secure tunneling capabilities. Most MPI implementations nowadays rely on ssh for remote startup of processes on nodes. It is natural to turn it to be VO-aware, i.e., to allow VO users interactively access remote XtreemOS nodes, provided that they are granted with appropriate rights.

As a demonstration and proof-of-concept of our approach, we modified OpenSSH to use the newly designed PAM/NSS extensions to authenticate VO users, namely XOS-OpenSSH. OpenSSH is a PAM-aware application that makes itself easily configured to use `pam_xos`. Also, `getpwnam` and `getgrnam` functions are called by OpenSSH to check if a user/group exists, which can be handled by the XtreemOS NSS module. Currently, we still need to modify source codes of OpenSSH due to the fact that user credentials need a special handling [2], though there are few lines of code to be changed.

For VO users to use XOS-OpenSSH, the only thing to provide to `xos-ssh` (the ssh client) is the credentials (XOS-Cert). Then, we are able to access any nodes that participate in the same VO, and that are equipped with `xos-sshd` (the ssh server). Also, remote command execution and third-party data transferring could be securely done within VOs. It is also possible to achieve "*zero*" configuration for VO users (e.g. for XOS-OpenSSH users, there is no need to input password or config traditional public-key pairs in both parties).

6.2 Interfaces for Node Administrators

There are several important configuration files that node administrators may be concerned with (in the worst case, no configuration means no allowed access for VOs). A conceptual list of them are as follows (detailed documentation is provided with the source code releases):

- For a PAM-aware application to use `pam_xos`, a configuration file needs to be put into `/etc/pam.d/`.
- For an application to do naming resolve between VO user ids/attributes and local uid/gid(s), `/etc/nsswitch.conf` needs to be modified to add the XtreemOS NSS module.
- The node administrator needs to configure local policies for VO user access, as discussed in Section 5.

[2]Currently proxy certificates are taken as user credentials, which need to be delegated from ssh client to ssh server

    - whether VO users are granted to access this node (by specifying mapping rules as well)
    - what access rights VO users will have for a set of local resources
    - what resource usage constraints to be put on VO users

There are command-line tools for node administrators to manage runtime user mappings and change local policies for active VO users.

6.3 Interfaces for Developers

Writing a PAM-aware application to `use pam_xos` To use `pam_xos`, the standard rules for writing a PAM-aware application are applicable, as documented in the Linux PAM page (*http://www.kernel.org/pub/linux/libs/pam/*). To pass user credential data from a application to the PAM module, PAM conversation functions should be used. It is an application-defined callback to allow a direct communication between PAM module and the PAM-aware application. In a parameter of `pam_start` function with the type of struct `pam_conv`, there is a field named `appdata_ptr` used to pass arbitrary data to the PAM module. A full example (simulating VO-aware `su`) can be found in the source distribution.

Get the user mapping information with NSS APIs Standard `libc` APIs, `getpw*` and `getgr*` could be used to get mapping information between VO users/groups to local uid/gid(s), when the local system is configured to use the XtreemOS NSS module. A detailed documentation about NSS is available at *http:// www.gnu.org/software/libc/manual/*.

Manipulation of user mappings with AMS APIs AMS client APIs can be used to perform the task of user mappings. Main related functions are:

- *amsclient_usermapping* Communicates with AMS server to accomplish a whole mapping process:
    - Input: VO user's identity and attributes
    - Output: Mapped user information in struct `GPASSWD` and `GGROUPS,` where `GPASSWD` contains the dynamically allocated local user information, and `GGROUPS` contains a list of local groups.
- *amsclient_clearmapping* To clear mapping data of a given VO user:
    - Input: VO user's identity or mapped local user identity

  - Output: none[3]

Fetching VO users' security token from the keyring Once granted access to the local node, VO user's security token is stored in kernel keyring which can be manipulated by KKRS APIs. The KKRS provides three new system calls to manipulate keys in user-space: `add_key` to create keys, `request_key` to search a process keyring for a key, and `keyctl` to perform various operations for managing keys. More information is available at *http:// lxr.linux.no/source/Documentation/keys.txt*. The simplest way to fetch VO user information from the keyring is to use wrapped APIs, such as:

- *xos_KRS_fetch* - to get stored security token from current process:
  - Input: buffer to hold fetched data, a description key with the format of `x509uk_xos_uid`, in which `uid` is the current local user id.
  - Output: the input buffer will be filled with VO user's credential (a proxy certificate)

Local policy management Local policies can be changed and made effective at runtime. Currently this part of the interface is under development and subject to change in the future.

A policy rule is defined in a multi-purposed `rule` struct, which is capable of accommodating different types of policy rules: user mapping rule, access control rule, resource usage rule and application specific policy rule:

```
struct rule_t
{
    rule_type_t type;  /* rule type:
                  MA - Mapping
                  AC - Access Control
                  RU - Resource Usage
                  AP - Application Specific
               */
    subject_t   subject; /* subject applied by the rule */
    object_t    object;  /* object applied by the rule */
    union
    {
        permission_t permissions;  /* for AC rule */
        quota_t      quota;       /* for RU rule */
```

[3]The return value only indicates success or failure, with setting `errno` to the failure cause.

```
        driver_t     ap_policy_driver; /* for AP rule */
        driver_t     ma_init_driver;  /* for MA rule */
    }
    rule;   /* the content of rule */
    /* indicate if the rule is disabled */
    bool        valid;
    /* indicate the period when the rule is applied */
    time_t      time;
    ...
}
```

The main related interfaces to set and enforce policies are (still under development):

- *xos_policy_add* Adds a new policy rule and puts it into effect:
  - Input: a policy rule
  - Output: a handle identifier.
- *xos_policy_change* Changes a policy rule to a new one and updates the runtime management system to keep up with this change. This interface is actually a composition of *xos_policy_remove* and *xos_policy_add*.
  - Input: a handle identifier, a new policy rule
  - Output: none
- *xos_policy_remove* Removes a policy rule and restores object status that affected by the rule before.
  - Input: a handle identifier.
  - Output: none

Other interfaces include enumerating current active policies or temporarily enabling/disabling a policy.

7 Conclusion

To provide native support for dynamically changing VOs in a secure, scalable and flexible way, we have exploited several existing mechanisms to extend user and resource management in Linux nodes to a VO scale. These mechanisms include

PAM, NSS and KKRS, with which VO user identities and policies can be mapped to local ones that are fully recognized by the Linux OS, in a quite transparent way for applications.

Without operating system support, it is difficult to differentiate access controlling and auditing among different VO users' accessing the same node, in a scalable and flexible manner, as well as preventing one user's applications from interfering with the others in terms of performance and other QoS constraints.

OS-level lightweight isolation and enforcement mechanisms, which are already in or approaching the mainline kernels, will be investigated and evaluated to be used for better support of VOs. These mechanisms include security enhancements such as Linux Security Module (LSM) framework and Security Enhanced Linux (SELinux), and virtualization techniques such as process containers and paravirtualization. Using those approaches, enforcement of VO policies at a fine-grained level, as well as efficient and thorough isolation of VO accesses, could be achieved in the local node. Typically these enhancements are done in an incremental manner so as to preserve the semantics and the API provided to normal Linux applications, ensuring backward compatibility.

Bibliography

[1] Kerberos. http://web.mit.edu/Kerberos/.

[2] Kernel based virtual machine. http://kvm.qumranet.com/.

[3] Kernel key retention service. http://lxr.linux.no/source/Documentation/keys.txt.

[4] Openvz. http://openvz.org/.

[5] Process containers. http://lwn.net/Articles/236038/.

[6] System databases and name service switch. http://www.gnu.org/software/libc/manual/html_node/Name-Service-Switch.html.

[7] XtreemOS consortium. D2.1.1: Linux XOS Specification. November 2006.

[8] XtreemOS consortium. D3.5.4: Second Specification of Security Services. November 2007.

[9] M. Erdos and S. Cantor. Shibboleth-Architecture DRAFT v0.5. http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-architecutre-05.pdf.

[10] Ford W. Housley R., Polk W. and D. Solo. [RFC 3280] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. 2002.

[11] Barham P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebar, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[12] V. Samar. Unified login with pluggable authentication modules (PAM). *Proceedings of the 3rd ACM conference on Computer and communications security*, pages 1–10, 1996.

[13] S. Smalley, C. Vance, and W. Salamon. Implementing SELinux as a Linux Security Module. *NAI Labs Report# 01*, 43, 2001.