Project no. IST-033576

XtreemOS

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

Linux-XOS for MDs/PDA
D2.3.4

Due date of deliverable: May 31st, 2008
Actual submission date: May 30th, 2008

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

Type: Deliverable
WP number: WP2.3
Task number: T2.3.4

Responsible institution: Telefónica I+D
Editor & and editor's address: Luis Pablo Prieto
Telefónica I+D
Parque Tecnológico de Boecillo
47151 Boecillo (Valladolid)
SPAIN

Version 1.01 / Last edited by Luis Pablo Prieto / May 28rd, 2008

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	1/4/08	Luis Pablo Prieto	TID	Initial template
0.2	3/4/08	Luis Pablo Prieto, Daniel Galindo, Manuel Martín	TID	Detailed outline
0.3	22/4/08	TID Team	TID	First draft of VO support chapter
0.4	24/4/08	TID Team	TID	First draft of Mobility chapter
0.5	24/4/08	Luis Pablo Prieto	TID	Introduction and roadmap
0.6	25/4/08	Luis Pablo Prieto	TID	Proofreading and re-structuration
0.7	28/4/08	Luis Pablo Prieto	TID	Final proofreading and formatting
0.8	22/5/08	TID Team	TID	Incorporated reviewer's comments
1.0	26/5/08	Luis Pablo Prieto	TID	Finishing touches
1.01	28/5/08	Luis Pablo Prieto	TID	Minor formatting issues

Reviewers:
Haiyan Yu (ICT), Antoine Giniès (EDGE)

Tasks related to this deliverable:

Task No.	Task description	Partners involved°
T2.3.4	Implementation of a basic Linux version for mobile devices	TID*, INRIA

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

Executive Summary

Mobile access to grid services is a research field that has not yet been fully solved, specially when it comes to the real implementation of such access. This has been mainly due to the heaviness of most grid service clients and middleware, which is not adequate for the restricted execution environment of mobile devices.

The XtreemOS project proposes to integrate several common grid functionalities into the operating system, to achieve better efficiency, scalability and transparency when operating inside virtual organizations, which would be used through similar methods as a usual operating system. This enhanced operating system will be developed for a number of platforms, including not only PCs but also clusters of computers and mobile devices, such as PDAs (XtreemOS-MD).

Towards this goal, the Linux operating system has been modified and extended with a number of software modules and services, which are broadly organized in two layers: a lower-level foundation layer (XtreemOS-F), which extends the operating system with grid entities so that virtual organization policies can be enforced, and a grid services layer (XtreemOS-G) which provides the grid services themselves, like execution or data management.

Xtreemos-MD, as the mobile device flavour of XtreemOS, will allow for mobile access to the XtreemOS grid services from Linux PDAs thanks to its foundation layer which provide virtual organization (VO) support. The first version of XtreemOS-MD has been constructed around the Ångström Linux distribution, although great emphasis has been made on the portability of the modules to other distributions. In fact, all the developed software can also be run over the Maemo distribution for Nokia Internet Tablets.

This document contains the installation, configuration and usage documentation of the foundation layer of XtreemOS-MD. This layer, apart from the aforementioned VO support in the Linux operating system, also provides terminal mobility features (e.g. the ability to change access networks without interrupting the connectivity) through a USAGI MIPv6 implementation for PDAs.

The software is organized into two packages, which can be installed independently, to accomodate the needs of different kinds of users:

The `xosmd-vosupport` package provides a flexible way for Linux to operate inside an XtreemOS virtual organization. Depending on the needs of the user and/or the Linux distribution upon which it will be integrated, several modes of operation are supported. The package includes a number of utilities in order to ease and automate the acquisition and management of credentials for grid access by user applications, as well as several mechanisms for safely storing these credentials to get single sign-on features in XtreemOS grids. Depending on the features of the mobile Linux distribution underneath, these utilities may depend on the mechanisms (namely, PAM and NSS) developed for the PC flavour of XtreemOS (e.g. if we want to isolate local processes from grid-related processes).

The `xosmd-mobility` package provides the terminal mobility features, and it is mainly composed of kernel-space software (which is already part of the mainline Linux kernel), plus a user-space daemon and a number of user tools.

Together, these two software packages provide the necessary low-level capabilities to seamlessly access XtreemOS grid services from a mobile Linux operating system, while moving from one access network to another, and to authenticate users against a virtual organization.

Contents

List of Figures

Glossary

AMS	Account Mapping Service
ARM	Advanced RISC Machine
ARMEL	ARM Emulator
API	Application Programming Interface
CN	Correspondent Node
FSUID	File system UID
GNOME	GNU Network Object Model Environment
HA	Home Agent
ICMP	Internet Message Protocol
IKE	IPSec Key Exchange
IPsec	Internet Protocol security
IPv6	Internet Protocol v6
KRS	Key Retention Service
MD	Mobile Device
MN	Mobile Node
MIPv6	Mobile Internet Protocol v6
NSS	Name Service Switch
PAM	Pluggable Authentication Modules
PDA	Personal Digital Assistants
PEM	Privacy Enhanced Mail
PKCS	Public-Key Cryptography Standards

RADVD	Router Advertisement Daemon
SA	Security Association
SAGA	Simple API for Grid Applications
SSH	Secure Shell
SSL	Secure Socket Layer
SVN	Subversion
TCP	Transport Control Protocol
UID	User Identifier
USAGI	UniverSAl playGround for Ipv6
VO	Virtual Organization
VOM	Virtual Organization Manager
WP	Work Package
XtreemOS-MD	XtreemOS for Mobile Devices

Chapter 1

Introduction

Mobile access to grid services is a research field that has spawned a number of publications and projects [5, 6, 8], but whose key issues have not been yet fully solved. Most (if not all) of them have proposed middleware-based solutions in service-oriented form, and have encountered several problems in their implementation phase, due to the inefficiency of such protocols, specially in restricted execution environments such as mobile devices.

In XtreemOS project a different approach has been proposed, which is to *integrate* several of the common *grid functionalities into the operating system* (more concretely, into a Linux operating system), to achieve better efficiency and scalability, and to increase the transparency of the grid, which would be used in the same way as if it were a usual operating system.

In order to achieve this goal, it is necessary that the operating system "understands" grid entities such as users or resources, in order to operate with them and enforce the Virtual Organization policies. In XtreemOS-MD, as in the other flavours of XtreemOS, this is achieved by the so-called *foundation* layer (XtreemOS-F, for short). This layer uses standard Linux methods like Pluggable Authentication Modules (PAM) or Name Service Switch (NSS), to incorporate grid users into the operating system's workflows.

Moreover, in the mobile device version of XtreemOS, a number of modifications have been made to this foundation layer to make it more usable and efficient, and an additional software package has been added to it, to provide enhanced terminal mobility using Mobile IPv6 protocols.

This document describes the first basic implementation of the foundation layer of XtreemOS-MD, and also details the processes that should be followed to build, install and run it.

The first version of XtreemOS-MD is based around a mobile Linux distribution for PDAs called Ångström [1]. This distribution has been chosen because it provides a good balance between modern software (as it would not make sense to build research prototypes on outdated software) and wide hardware support (one of the main problems of most embedded Linux distributions). However, in the

development of the software great emphasis has been put in the *portability* and *flexibility* of the code presented here, which should be integrable into any modern mobile Linux system, regardless of the distribution used. In fact, the code has been successfully integrated into Nokia Internet Tablets running Maemo Linux OS.

1.1 XtreemOS-MD usage scenario

To better illustrate the capabilities of the software presented in this document, and show how the basic version of XtreemOS-MD could be used, please imagine the following scenario[1]:

> Jane Doe has just started working in an aerospace company, in a project to design the wings of a passenger airline. She is in charge of supervising the 3D models of the wing, which must comply with a number of aerodynamic requisites, and to discuss with their providers about the feasibility and details of obtaining the pieces of the wing. As she will be travelling to the providers' factories and offices to talk with them, she has been given an internet tablet, so that she can work while being on the move.
>
> Just as she is heading to her first meeting with the providers, she powers up the tablet and discovers that it is an out-of-the-box device, with nothing installed there that could allow her to work properly. Luckily, among the project's documentation there is a description of the applications they use for doing the modelling and simulation (using a grid computing facility shared by the company and the providers, which uses something called XtreemOS for its infrastructure). It also describes that a software repository for mobile devices is available for her to install all the necessary software.
>
> Following the instructions, she just configures the software repository in the tablet's software installation application, and installs the basic software for operating with XtreemOS. She also installs a small application to check the status and results of the simulations that are running in the company's grid, as well as a popular instant messaging application to securely chat with providers and with other members of her team.
>
> Once she has installed everything (without being an expert in grid technologies or mobile devices), she starts the application to check the simulations. The application asks for a user and password in order to get some credentials from a security server and securely access her simulations. After entering the correct username and password, she

[1] Please bear in mind that the software supports many other modes of working, this is just one possible scenario.

is presented with a list of all her simulations. She sees that the desired simulation has finished, and visualizes the output data, which is securely stored in some backup machine in the grid (without asking again for username and password to access the data).

However, Jane notices that some error has ocurred, and she decides to talk with one of the providers to discuss the issue. She fires the chat application, which looks and behaves like the PC application that she has always used in her desktop. This time, no questioning about username and password takes place either, and a secure channel is established for them to chat about the problem. During the conversation, she is able to access all the logs from past conversations with the provider. In the end, it seems that the input data was erroneous, and Jane launches a new simulation from her tablet, this time with the correct data, and just in time for the meeting.

In this scenario, both the applications and the grid services that provide the certificates and the execution and files access are software modules that depend on the scenario's virtual organization (the aerospace company and its providers), and thus are out of the scope of this document. However, the underlying mechanisms for requesting, storing and reutilizing Jane's certificates are the focus of the software presented here.

Also, thanks to the terminal mobility features of XtreemOS-MD (which are also described in this document), all the aforementioned operations could be seamlessly performed while moving in the taxi, jumping from hotspot to hotspot, or using the device's cellular connectivity.

1.2 XtreemOS-MD architecture

Figure 1.1 shows the latest version of the software architecture of a XtreemOS-MD node.

As explained in earlier project documents [12], the architecture is composed of a number of standard software modules that are present in any mobile Linux distribution (kernel, standard libraries, graphic user interface etc), plus a number of XtreemOS-specific software modules, which can be grouped into two main layers:

- The foundation layer (F-layer) we already mentioned, mainly providing OS-level support for Virtual Organizations and terminal mobility support.
- The grid services layer (G-layer), which provides the grid functionality itself, like execution management, data management or security features, plus a standard API for grid applications.

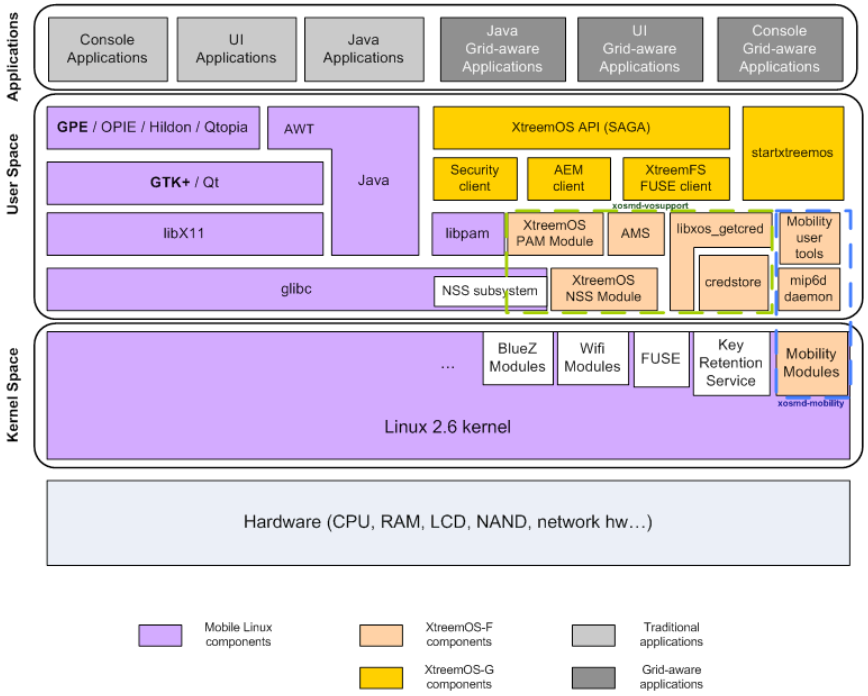


Figure 1.1: XtreemOS-MD Software Architecture

1.3 F-Layer modules

The F-layer of XtreemOS for mobile devices is composed of two main software packages:

`xosmd-vosupport` provides Virtual Organization support in Linux for mobile devices, in a similar fashion as the package `xos-nss-pam` does for the PC flavour [11]. This includes grid authentication in local nodes using PAM, grid users information interface through NSS, plus additional modules to make it all work, like a common interface for accesing credential storages (`libxos_getcred`), various methods of storing user credentials (`credstore`, or the kernel's Key Retention Service), and the (optional) local-VO identities mapping
database (Account Mapping Service – AMS).

`xosmd-mobility` provides terminal mobility features for XtreemOS-MD, by implementing Mobile IPv6 protocol [7], based on the USAGI [10] implementation of the protocol. This software module comprises both a number of kernel patches (to enable MIPv6 low-level features), a daemon (`mip6d`) for sending and receiving specific protocol messages, and the corresponding user tools.

1.4 Document structure

This document is structured as follows:

Chapter 2 describes how to build, install and use the VO support features of XtreemOS-MD. This includes the system requirements, the installation process, the steps to correctly configure the utilities and the user's manual and APIs that this module offers.

Chapter 3 describes the features of the terminal mobility software package, including system requirements, installation, configuration and user tools that this package comprises.

Chapter 4 explains the main future milestones of the development of XtreemOS-F for mobile devices.

Finally, an appendix includes information that can be useful to XtreemOS-MD and application developers: a description of the XtreemOS-MD development environment.

Chapter 2

VO Support in Linux-MD

2.1 Main features

In the basic version of XtreemOS-MD, mobile devices will act only as grid resource consumers (e.g. launching grid jobs, accessing grid files etc), providing lightweight access to the XtreemOS grid. Thus, it is not absolutely necessary to implement the same VO support mechanisms as for the PC flavor of XtreemOS, which is targeted at PCs that do share their resources in a virtual organization.

The main goal of the software presented here is to make the access of client-side grid applications (specially mobile applications) easier and more transparent. This is done by providing a flexible framework for managing XtreemOS credentials in a mobile node (to implement **single sign-on** features), and by including a number of mechanisms for **transparently** acquiring and managing these credentials, both for **XtreemOS-aware** and **legacy** (XtreemOS-unaware) applications.

On the other hand, since different mobile users have different ways of working, and there could be users who want some degree of **isolation** between their grid-related applications and non-grid applications, some of the mechanisms for **mapping grid and local users** developed for the PC flavour (see D2.1.2 [11]) have also been adapted to mobile devices. This also has the advantage of providing a first stepping stone in the way to sharing the resources of mobile devices with the grid (an option that will be investigated in the advanced version).

Apart from these features, the software described in this chapter has also been implemented with **portability** in mind, both between hardware platforms and between different Linux distributions. The **platform portability** is important since most of the enhancements presented here are also useful in a PC or even in a cluster environment, and as such they are being ported to the other XtreemOS flavors. Linux **distribution portability** is also important because the fragmentation of the Linux (and specially *mobile* Linux) market makes distribution-agnosticism very advisable for exploitation purposes.

This software also supports **internationalization**, and thus all the messages and user interactions can be easily translated to other languages.

Finally, it is also worth noting that, although these software modules have been designed with XtreemOS in mind, the mechanisms could also be used by any other system that uses PKI credentials, with minor modifications.

2.2 Software description

The VO support funcionalities of XtreemOS-MD are grouped in two software packages:

- `xosmd-vosupport` provides the minimal VO support features necessary for using a mobile Linux device as a XtreemOS grid client. This package includes:
  - A set of launcher applications designed to transparently perform all the necessary operations for initializing XtreemOS VO support features in a Linux distribution, under a number of VO and mobile device configurations.
    - `startxtreemos-m` is the simplest implementation, which does not rely on external software and does not isolate grid applications. Ideal for systems that do not support PAM modules (i.e. `libpam`) and do not require special isolation for grid-related applications.
    - `startxtreemos` relies on the XtreemOS PAM module (i.e. `xos-nss-pam` and `libpam`) to perform the authentication, but does not provide any isolation for grid-related applications.
    - `startxtreemos-ams` provides isolation for grid-related processes by mapping the current user session to a new (temporary) local identity generated on the fly, mapped to the grid identity represented in the XtreemOS credentials. Requires the `xos-nss-pam` and `libpam` packages.
  - A generalized abstraction of a credential store to implement single sign-on features in XtreemOS-MD and the necessary API to manipulate these stores, to be used by end user applications (`libxos_getcred`) and by other XtreemOS components (`libcredstore`).
  - A minimal but complete credential store implementation that does not depend on any external software (`uskeystore`).
  - Another implementation of a credential store, using the Linux kernel Key Retention Service (`krs`).
  - A way for legacy (XtreemOS-unaware) applications to get access to these credential stores (`libxos_wrapopen`).
  - Additional graphical applications for mobile users to interact in the credential management process (`read_protected_credfile`).

- `xos-nss-pam` is a porting of the VO support features available in the PC flavor of XtreemOS. With this package installed, XtreemOS-MD users obtain:
  - Credentials-based authentication using the XtreemOS PAM module (which, in turn, depends on `libpam`).
  - A certain level of isolation between grid-related and ordinary, local processes, through the mapping of the user's grid identity to a different local identity.

Since the `xos-nss-pam` features and software components (namely, the XtreemOS NSS and PAM modules, and the AMS daemon) are already explained in the PC flavour user manual (see D2.1.2 [11]), we will concentrate here on the `xosmd-vosupport` features and documentation, with mobile-specific references to `xos-nss-pam` where appropriate (e.g. to detail the installation procedure of this package in a mobile device).

In the following sections we will describe how all these elements work together to attain the aforementioned VO support features, and we will also describe each of them in detail.

2.2.1 Typical usage scenarios

Figure 2.1 shows how these elements work together in the scenario of a XtreemOS-aware application which requests a grid user credential (in XtreemOS called *XOS-Cert*) for later access to any XtreemOS grid service. In this case, the application does not know (or care) whether the credentials are already available locally or not but, in the end, the credentials are delivered, and they will be stored locally in case the same or other application or grid service needs them.

Figure 2.2 shows the inner workings of the scenario where the application does not even know any grid-related library. In this case, the credential store can be accessed via simple file opening commands, through the `libxos_wrapopen` library.

And finally, figure 2.3 shows how further process isolation can be attained through the usage of the VO support mechanisms available in the `xos-nss-pam` package. In this case, the user must explicitly start the `startxtreemos-ams` helper, since it must be executed with the *setuid* `root` attribute. A session (or an application) with a new, temporary local user is started (mapped to the grid identity provided by the XOS-Cert), and all subsequent applications and grid services are executed with that new local identity.

2.2.2 `libcredstore` and `libxos_getcred`

A **credential store** (credstore for short) is a mechanism for storing/retrieving a credential (e.g. a X.509 certificate like the XOS-Certs XtreemOS uses), with support for establishing credential timeouts or manually deleting the credential. The

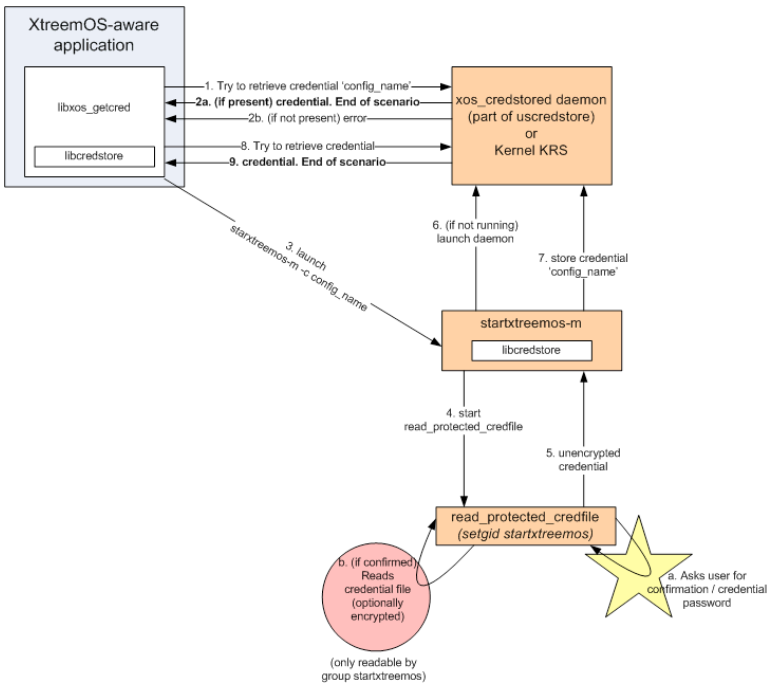


Figure 2.1: Acquisition of credentials by a XtreemOS-aware application

current `libcredstore` library is intended for **abstracting the credstore functionality** from the concrete implementation, and allows for several different mechanisms for the implementation of the credstore. Right now, two implementations are provided with the software, one based on the Linux Kernel Key Retention Service (KKRS) (`krs`, with variations for storing credentials in compressed format – `zkrs`) and a user space daemon implementation (called `uskeystore`).

Each user in the system has its own user credstore and may also have several session credstores. A **user credstore** can be accessed by all the processes executed by a user, while a **session credstore** can be accessed only by one concrete process of a certain user (the typical example being a command line shell), and all its children.

Currently, each credstore can store only one credential[1], since supporting multiple credentials is against the philosophy of single sign-on security. However, we are aware that, under different configurations, the user could have to manage different identities or credentials, and thus labelling of credentials with a **configuration name** is supported. If an application asks for a credential from the credstore, it may check that the configuration name of the stored credential is the one expected by the application. Configuration name "default" is assigned when a new credential is stored without specifying a configuration name.

[1]From now on, we will denote with the word "credential" the set of objects stored in the credstore, which are the XtreemOS certificate (with the public key) and the corresponding private key.

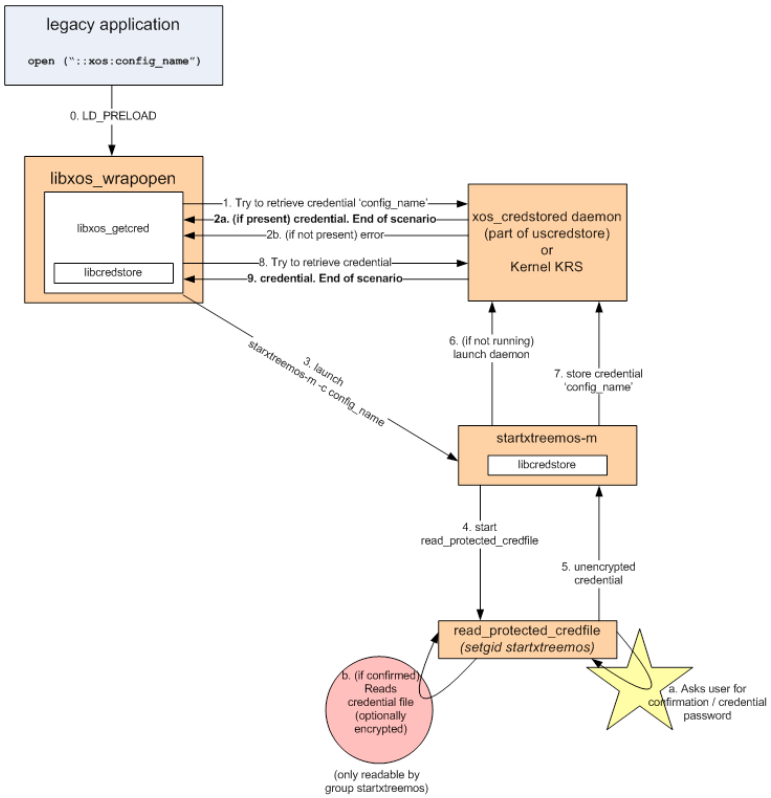


Figure 2.2: Acquisition of credentials by a legacy (XtreemOS-unaware) application

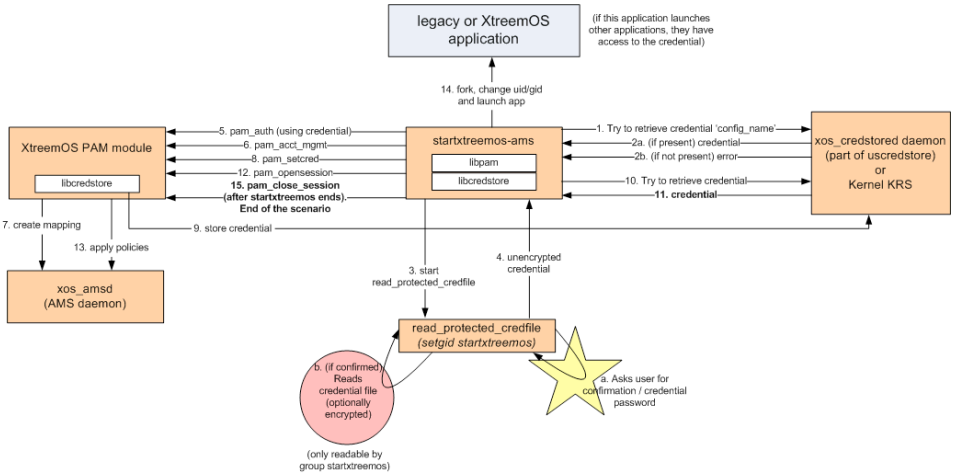


Figure 2.3: Acquisition of credentials with enhanced process isolation

libcredstore

libcredstore is a **credential storage abstraction** library. The goal of a libcredstore is to provide a credentials cache, in order to implement a single sign-on mechanism, independently of the real storage system used (e.g. the Kernel Key Retention Service). libcredstore abstracts the use of the Key Retention Service and allows for the use of other replacements without modifying or recompiling applications. This feature is very important in embedded devices, as recompiling and replacing the kernel may be very complex for non-expert users (i.e. the Key Retention Service is generally not compiled by default and cannot be installed as module). For example, in Maemo Nokia devices (e.g. N800 and N810), users can install applications with a click, but to support KRS it requires that the user reflashes the device to change the kernel, with the subsequent risk of making the device unusable (also known as "bricking" the device).

libcredstore uses the Key Retention Service by default if it is supported by the runtime kernel, with a new feature: it compresses and decompresses keys on the fly. It does not require that the libkeyutils package is installed, because it invokes syscalls directly. If the kernel does not support KRS, libcredstore detects that the syscall fails and uses a replacement consisting in daemons executed in user space (e.g. uskeystore).

The libcredstore API is not needed by programs that use XtreemOS: these applications will use libcredstore indirectly, either through the libxos_getcred API, the PAM module API or higher level APIs like SAGA (which would invoke libcredstore internally). Thus, end user applications should not be linked with libcredstore. Instead, they should be linked with libxos_getcred or libpam (or the corresponding SAGA equivalent). Client applications that need to obtain a credential use libxos_getcred, and node resource applications that need to authenticate a user through its XtreemOS credential use libpam.

The functionalities covered currently in libcredstore are reading, writing and deleting a credential, and it also allows setting a timeout after which the credential is purged from the credstore.

Further documentation about libcredstore and its associated tools is present in the "Command line tools" section (2.6). These tools are replacements of keyctl, the command line utility provided with the KKRS implementation in the PC flavor.

libxos_getcred

In order to ease the labors of **end user application** developers, a simpler interface has been implemented for retrieving credentials, relieving the developer of needing to know the credstore's API and the protocols that should be followed to use it correctly. This library (libxos_getcred) makes the adequate calls to the libcredstore API in order to obtain the credential, and launches the available version of startxtreemos to obtain the credential if the credential is not

present in the credstore.

The aim of this library is to ease the implementation of a caching single sign-on system. It provides a method for applications to obtain a XtreemOS credential (although the mechanism is generic, and can be used with other types of credentials) and ensures that this credential is cached to avoid disturbing users by asking them for a password every time an application needs to use the credential.

Any application using it just needs a simple library call for obtaining the credential, which will be cached in a credential store (credstore), e.g. to avoid asking every application for passwords in case they would like to use the key. When an application uses this library, it obtains the current credential in the credstore (if it is not empty and the configuration name of the stored credential is the same as the one required). Otherwise, `startxtreemos` (see below) is launched automatically.

2.2.3 New client software: `startxtreemos*`

Client grid nodes need a method for user applications **to obtain a XtreemOS credential** (although the mechanism is generic, and can be used with other types of credentials), as well as **to ensure that this credential is cached** (using `libcredstore`) to avoid disturbing users, asking them for a password every time an application needs to use the credential (i.e. to provide single sign-on).

The current implementation in XtreemOS-MD offers two methods to start a XtreemOS session, by obtaining a user credential, which is then shared with other applications using the credstore:

- Explicit start of the XtreemOS session, by invoking one of the `startxtreemos` tools.
- Automatic, on-demand startup of the XtreemOS session, when an application demands a credential[2]. If the credential is not present in the credstore or the credential is labelled with a configuration name different from the one requested by application, `startxtreemos` is invoked automatically.

The explicit call to `startxtreemos` allows for advanced operations like, e.g. to start a new session (with its own session credstore), run a program in its own session or set a timeout for the session.

There are three versions of the `startxtreemos` tool, depending on the kind of usage that the user (and the virtual organization under which he operates) requires from XtreemOS-MD:

- `startxtreemos-m` is the simplest implementation, which does not rely on any external software and does not isolate grid applications from other user applications (i.e. grid applications are started using the same local UID that was being used). This implementation is recommended specially for

[2]The act of demanding a credential can be achieved both by XtreemOS-aware (by using the `libxos_getcred` library) and legacy applications (by opening a file called "::xos:config_name").

embedded systems, that do not support PAM modules (i.e. `libpam`) and do not require special isolation for grid-related applications.

- `startxtreemos` provides the same functionality as the `startxtreemos-m` version, but it relies on the XtreemOS PAM module (i.e. `xos-nss-pam` and `libpam`) to perform the authentication.

- `startxtreemos-ams`, additionally, provides isolation for grid-related processes by mapping the current user identity to a new (temporary) local identity generated on the fly, corresponding to the grid identity represented in the XtreemOS credentials. This version requires the `xos-nss-pam` and `libpam` packages.

In order to obtain the credential that corresponds to the specified configuration name (e.g. "config_name"), the current implementation searches for a `<config_name>.pem` file in the `/etc/xos/creds/` directory. This is just a proof of concept until a proper mechanism for obtaining the credential (e.g. by connecting to the Credential Distribution Agency – CDA) is developed in the XtreemOS-G layer.

Since this way of working would be insecure (if any user can read directly `/etc/xos/creds`, the credstore's security is useless), a more secure implementation option is provided. If the software is compiled with the `USE_READ_PROTECTEDCRED_HELPER=1` option set in the Makefile, then a more advanced mechanism is used to read the credential, based on a **helper GTK+ program** (`read_protected_credfile`).

When using this mechanism, `/etc/xos/creds/` and all the credentials it contains, is only readable by the "startxtreemos" group. However, the helper program has the appropriate *setgid* attribute, and can read this directory. The helper program asks for user confirmation to read the file (in case the process was not started by a user application but by some other malicious application), reads it and returns it to `startxtreemos`. This helper program is invoked only to read credentials from `/etc/xos/creds`, not for reading credentials from the credstore.

Optionally, the credential files in `/etc/xos/creds` can be encrypted. These files are currently denoted with the ".crypt" extension instead of ".pem". In this case, the helper program asks the user for the passphrase to decrypt the key. The user is not asked again about the passphrase while the credential is in the credstore, since the credential is already stored in the credstore in unencrypted form.

This helper program is configurable using the `/etc/xos/read_protected_credfile.conf` file.

2.2.4 Mechanisms for legacy applications: `libxos_wrapopen`

Since one of the main goals of XtreemOS-MD (and of XtreemOS in general) is transparency for applications trying to use the grid, even for legacy applications, the VO support features of XtreemOS-MD include a mechanism for these

XtreemOS-unaware applications to **access the credentials** that will enable them to later access the XtreemOS grid services. This mechanism is implemented in the `libxos_wrapopen` library.

This library uses a procedure which is analogous to the one used by applications using SOCKS, through the usage of the `LD_PRELOAD` environment variable. This variable indicates a library which will have precedence over any other library (`libc` included), effectively overriding the `open()` call for opening files. In the current implementation, just invoking `open()` with "::xos:config_name" as a parameter, will try to fetch (through the usual `libxos_getcred` interface explained above) the credential whose configuration name is "config_name".

2.3 System Requirements

2.3.1 Hardware

The compiled binaries available from the repositories run in any ARMv5 or upper ARM processor, with at least 16 MB of RAM. They are also able to run in a QEMU emulator that may run in any modern PC with a PentiumIII/IV/Celeron processor and at least 512 MB of RAM.

2.3.2 Software

The software requirements of the VO support functionalities ported from the PC flavor (`xos-nss-pam`) are described in D2.1.2 [11]. The software requirements of the different components of the `xosmd-vosupport` package are:

- `libcredstore` only depends on `libz-dev`. This dependency is needed only at build time, not at runtime, because the library is compiled with the static version of `libz` library.
- `libxos_getcred` and `startxtreemos-m` do not have any dependencies. But if the project is compiled with the option `USE_READ_PROTECTEDCRED_HELPER=1` in the Makefile (i.e. read_protected_credfile is built), then `libgtk+` 2.12 or upper and `libcrypto` 0.9.7 or upper (normally part of the `openssl` package) are required. The Maemo porting of the software also depends on the Hildon libraries.
- `starxtreemos` and `startxtreemos-ams` depend on `libpam`. At runtime, they use `xos-nss-pam` source code (XtreemOS PAM and NSS modules, `xos_amsd` daemon, `libxos_db`, `libxos_security`, `libxos_ams`, `libxos_plymgt`, `libopenssl`/`libcrypto` 0.9.8 and `libdb` 4.3).

Caveats and known issues

At runtime, applications with `l12n` files running in Ångström, messages do not appear according to the locale if package `glibc-binary-localedata-*` is not installed.

For example, in order to see the messages in Spanish in Ångström, the command `ipkg install glibc-binary-localedata-es-es` would be needed.

2.4 Installation manual

Although all the different software components are currently part of the same software package (`xosmd-vosupport`), different users will have different needs with regard to the VO support and single sign-on features that they require. Depending on the concrete components that the user (or the Linux integrator or distributor) needs, different building and installation alternatives can be followed:

- If only the bare minimum functionality (the credstore abstraction) is needed, because all the other mechanisms are going to be implemented specifically for the concrete mechanisms of the VO/distribution, then only the `libcredstore` library must be built and installed (see section 2.4.1).
- If a lightweight but complete implementation of the single sign-on mechanisms and easy integration for end user applications is desired, then the `libcredstore` and `libxos_getcred` libraries, plus the `startxtreemos-m` launcher application, must be built and installed (see section 2.4.2). This is the recommended configuration for most cases.
- If a complete implementation of the single sign-on mechanisms and easy integration for end user applications is needed, and isolation of grid-related and local processes is desired, then the `libcredstore` and `libxos_getcred` libraries, plus all the `startxtreemos*` applications, must be built and installed (see section 2.4.3). In addition, we also need to build and install the VO support features ported from the PC flavor (the `xos-nss-pam` package, see section 2.4.4). This option is experimental, and is recommended only for expert users.

The different installation alternatives are described below[3]:

2.4.1 Building and installing `libcredstore`

`libcredstore` is part of the `xosmd-vosupport` project/package. An `INSTALL` file in the project source code contains more detailed instructions, but the basic process is:

[3] Since the software is also being ported to Maemo (for Nokia tablets), instructions are also given where appropriate.

1. Install the `libz-dev` package.
2. Run `make install-credstore` as root or run `make package-credstore.tgz` and unpack the `package-credstore.tgz` file into your system, wherever you want to install the package.
3. The software is normally installed in `/usr/local`; it may be necessary to invoke `ldconfig`[4], or the addition of `/usr/local/lib` to `/etc/ld.so.conf`.

2.4.2 Building and installing `libcredstore`, `libxos_getcred` and `startxtreemos-m`

Currently, the source code of all these modules is part of the `xosmd-vospport` project/package. An `INSTALL` file in the project source contains more detailed instructions, but the basic process is:

1. Install the `libz-dev` package.
2. If we want to include `read_protected_credfile`, edit the Makefile and uncomment the line `USE_READ_PROTECTEDCRED_HELPER=1`. If building for Maemo, also uncomment `USE_MAEMO=1`.
3. Run (as root) `make install-m` or run `make package-m` and unpack `package.tgz` to the desired location to install the package and, after unpacking, run `./postinstall.sh`.
4. The application will look for credentials in `/etc/xos/creds`; you may copy the content of the `examplecreds` file in `/etc/xos/creds` (it contains some example credentials that can be used for testing the installation).
5. The software will be normally installed in `/usr/local`; it may be necessary to invoke `ldconfig`[5] or to add `/usr/local/lib` to `/etc/ld.so.conf`.

2.4.3 Building and installing `libcredstore`, `libxos_getcred`, `startxtreemos-m`, `startxtreemos` and `startxtreemos-ams`

Currently, the source code of all these modules is part of the `xosmd-vospport` project/package. An `INSTALL` file in the project source contains more detailed instructions, but the basic process is:

[4] That is, executing `ldconfig /usr/local/lib` as root.
[5] That is, executing `ldconfig /usr/local/lib` as root.

1. Install the `libz-dev` package.
2. Install the `pam-dev` package. This package is not included in Ångström (nor Maemo), see below for instructions about compiling the package.
3. If we want to include `read_protected_credfile`, edit the Makefile and uncomment the line `USE_READ_PROTECTEDCRED_HELPER=1`. If building for Maemo, also uncomment `USE_MAEMO=1`.
4. Run (as root) `make install-m` or run `make package-m` and unpack `package.tgz` to the desired location to install the package and, after unpacking, run `/postinstall.sh`.
5. The application will look for credentials in `/etc/xos/creds`; you may copy the content of the `examplecreds` file in `/etc/xos/creds`.
6. In `/etc/xos` there are examples of configuration files. The `/etc/pam.d` directory contains the required files to configure PAM programs. You may invoke `make install-all` instead of `make install` and `make package-all` instead of `make package` to also include the example configuration files and credentials.
7. The software will be normally installed in `/usr/local`; it may be necessary to invoke `ldconfig`[6] or to add `/usr/local/lib` to `/etc/ld.so.conf`.

2.4.4 Building and installing `xos-nss-pam` source code in a ARM machine

The process of building and installing the VO support code provided by the `xos-nss-pam` package in a mobile device is described below (including the support for `libcredstore`). Both instructions for Ångström distribution and Maemo (Nokia N800/N810) are provided. This PC flavor code is required by `startxtreemos` and `startxtreemos-ams` (but not for `startxtreemos-m` / `libxos_getcred`). It would also be necessary if the device runs as a grid resource node and uses the PAM module to authenticate grid users (a functionality currently not supported in XtreemOS-MD).

Detailed instruction for this installation is included in the `xos-nss-pam` source code (see `README.Angstrom` and `README.Maemo` files).

The process for building the code inside an Ångström distribution, for example using the QEMU-based development environment (see appendix A), is as follows[7]:

[6] That is, executing `ldconfig /usr/local/lib` as root.

[7] Currently, the Maemo binaries are also compatible with Ångström, but this is not guaranteed in future releases of Ångström and Maemo.

1. Install libdb and check

```
ipkg install db-dev check-dev
```

2. Install PAM. Unfortunately, the PAM package is not included in Ångström, so we need to compile it:

```
ipkg install flex-dev tar
wget http://www.kernel.org/pub/linux/libs/pam/library/Linux-PAM-1.0.0.tar.gz
tar xzvf Linux-PAM-1.0.0.tar.gz
cd Linux-PAM-1.0.0/
./configure CFLAGS=-fno-strict-aliasing
make install
```

3. The software also requires OpenSSL 0.9.8, but Ångström only provides OpenSSL 0.9.7. Thus, we need to download and compile OpenSSL 0.9.8:

```
ipkg install perl-module-integer
wget http://www.openssl.org/source/openssl-0.9.8g.tar.gz
cd /openssl-0.9.8g
./config --prefix=/usr/local shared
(remove optimizations (-O3) in the Makefile)
```

4. The software also depends on `libcredstore` (see above):

```
ipkg install libz-dev
(if you would like to compile read_protected_file helper):
ipkg install pkgconfig gtk+-dev renderproto-dev
cd xosmd-vosupport
make install-all
```

5. Finally, we will be able to compile the source code of the `xos-nss-pam` package. Since SVN source code of `xos-nss-pam` does not include a `configure` script, we must generate it using `autoreconf -install -force` *outside QEMU* (e.g. in a PC machine) and copying it afterwards.

```
./configure CFLAGS=-Wall
make install
```

6. Configure `/etc/nsswith.conf`. We must edit two lines: the lines that starts with `passwd:` and the line that starts with `group`. In both cases, append at the end of the line `xos`.

7. Create a `/etc/xos/cacert` directory and copy the file `./src/examples/testcerts/xosca/xtreemos-ca.crt` into it.

8. Edit `/etc/xos/pam-xos.conf` and change the value of option `VOCaCertDir` to `/etc/xos/cacert`.

9. The software normally will be installed in `/usr/local`. It may be necessary to invoke `ldconfig`[8] or to add `/usr/local/lib` to `/etc/ld.so.conf`.

[8] That is, executing `ldconfig /usr/local/lib` as root.

On the other hand, the steps for building the software in a Maemo 4 platform (Nokia N800 and N810 Internet Tables) is as follows[9]:

1. Start `scratchbox` and change to an ARMEL target:

```
sb-conf se CHINOOK_ARMEL
```

2. Download, compile and install `flex` (we need flex only to compile PAM; afterwards we may delete it safely):

```
wget http://prdownloads.sourceforge.net/flex/flex-2.5.35.tar.gz
tar xzvf flex-2.5.35.tar.gz && cd flex-2.5.35 && ./configure && \
make install
```

3. Download, compile and install PAM:

```
wget http://www.kernel.org/pub/linux/libs/pam/library/Linux-PAM-1.0.0.tar.gz
tar xzvf Linux-PAM-1.0.0.tar.gz && cd Linux-PAM-1.0.0/ && configure && \
make install
```

4. Download, compile and install OpenSSL 0.9.8 (the 0.9.7 version available in Maemo is not sufficient):

```
wget http://www.openssl.org/source/openssl-0.9.8g.tar.gz
tar xzvf openssl-0.9.8g.tar.gz && cd openssl-0.9.8g && \
./config --prefix=/usr/local shared && make install
```

5. Download, compile and install `check`:

```
wget http://ftp.debian.org/debian/pool/main/c/check/check_0.9.2.orig.tar.gz
tar xzvf check_0.9.2.orig.tar.gz && cd checkr-0.9.2 && ./configure && \
make install
cp /usr/local/share/aclocal/check.m4 /usr/share/aclocal/
```

6. Install `libz-dev` (needed to compile `libcredstore`):

```
apt-get install libz-dev
```

7. Download, compile and install `libcredstore` (which is inside package `xosmd-vosupport`):

```
cd xosmd-vosupport
make install-credstore
```

8. Download, compile and install `libdb-4.3` (or as an alternative, this package from the Ångström distribution):

```
wget http://download.oracle.com/berkeley-db/db-4.3.29.tar.gz
tar xzvf db-4.3.29.tar.gz
cd db-4.3.29
patch -p2 < patch-libdb4.3.29.oe.patch
cd build_unix
env CFLAGS=-Os ../dist/configure --enable-o_direct \
--disable-cryptography --disable-queue --disable-replication \
--disable-statistics --disable-verify --enable-compat185 \
--prefix=/usr/local/ --with-mutex=ARM/gcc-assembly
make
```

[9] In this case, Scratchbox and the Maemo SDK are needed. Please follow the instructions described in http://tablets-dev.nokia.com/4.0/INSTALL.txt

9. Finally, compile and install the source code of `xos-nss-pam` itself:

```
svn checkout \
https://scm.gforge.inria.fr/svn/xtreemos/WP2.1/T2.1.3/trunk/src/nss_pam/
cd nss_pam && autoreconf --install --force
./configure && make install
```

10. Install the software into a N800 machine (we will need root access with SSH). Create this package in scratchbox and decompress it in the N800:

```
tar czvf wp21-package.tgz /usr/local/bin/xos* \
/usr/local/lib/libcrypto.so.* /usr/local/lib/libssl.so.* \
/usr/local/lib/libdb-4*.so /usr/local/lib/libcredstore.so* \
/usr/local/lib/libnss_xos.so /usr/local/lib/pam_xos.so* \
/usr/local/lib/libxos_*.so /etc/xos/* /lib/libpam.so*
```

11. Configure `/etc/nsswith.conf`. In this case, we must edit two lines: the lines that starts with `passwd:` and the line that starts with `group`. In both cases, append at the end of the line `xos`.

12. Create a `/etc/xos/cacert` directory and copy the file `./src/examples/testcerts/xosca/xtreemos-ca.crt` into it.

13. Edit `/etc/xos/pam-xos.conf` and change the value of option `VOCaCertDir` to `/etc/xos/cacert`.

14. The software normally will be installed in `/usr/local`. It may be necessary to invoke `ldconfig`[10] or to add `/usr/local/lib` to `/etc/ld.so.conf`.

2.5 Configuration

2.5.1 `libcredstore`

The current `libcredstore` implementation does not have any configuration files. Credential files are expected to be in the `/etc/xos/creds` directory. The naming convention is that unencrypted files end with `.pem` extension, and encrypted files end with `.crypt`.

2.5.2 Client node modules

As an API, `libxos_getcred` does not require any special configuration. However, some comments must be made about an advanced feature of the library: aliasing.

[10] That is, executing `ldconfig /usr/local/lib` as root.

Aliasing in `xos_getcred()`

Applications may invoke `xos_getcred()` with a not-NULL configuration name. For example, each application may choose its own application name as the configuration name. But, if each application uses a different configuration name, the single sign-on mechanism will be useless, since each configuration name corresponds to a different credential (and the current implementation can only store one credential).

The solution is that the device administrator defines (in `/etc/xos/configname_alias`) that a group of configuration names are actually the same credential (i.e. it is an aliasing method). The syntax of the file consists of one or more lines, each one with a series of configuration names separated by "/". All names specified in the same line are considered aliases of the first name. When an application invokes `xos_getcred`, if a name is found in `/etc/xos/configname_alias`, it is replaced with the first name of the line in which it appears.

For example, with the line:

```
default/internet/mail/xmpp/xtreemos
```

If an application invokes `xos_getcred("xmpp")`, the configuration name used is "default" instead of "xmpp".

`read_protected_credfile` configuration

The `read_protected_credfile` program is configurable using `/etc/xos/read_protected_credfile.conf`. Current version supports four parameters in the security section:

forced_display this option is used to hardcode a value in the `DISPLAY` variable. The objective is to avoid that a malicious application changes the display to other machine in order to confirm that reading a file under `/etc/xos/` directory is authorized (instead of showing it to the user).

grabserver if this option is true, the GTK+ client will "grab" the X-Window server. This is a security measure, used to avoid that a malicious application reads the password, or sends an event to confirm a dialog bypassing user interaction.

autokill_after kills the program helper if the user does not respond after the specified number of seconds.

lockfile allows to specify a lock file to avoid that several instances of the program run simultaneously. If the system is multiuser, probably you should use a file under the user's home directory.

`startxtreemos*` configuration

Both `startxtreemos` and `startxtreemos-ams` use `xos-nss-pam` code and therefore they are indirectly affected by its configuration files, that are usually stored in `/etc/xos`. Please refer to the PC flavor VO support deliverables for more information [11].

2.5.3 PAM and NSS modules (`xos-nss-pam` package)

These modules have been originally developed for the PC flavor of XtreemOS. They have been ported to a mobile device architecture (ARM), and they are used by `startxtremos` and `startxtreemos-ams` (however, they are not used by `startxtreemos-m` or `libxos_getcred`). The configuration of these modules is covered in the PC flavor VO support deliverables [11].

2.6 Command line tools

2.6.1 `libcredstore` command line tools

The libcredstore library also contains some useful command line tools for easily accessing its functionality:

`xos_dumpcred [<configuration_name>]`
If no configuration name is specified, it returns the credential stored in the credstore (if any). If a configuration name is specified, first it checks that the configuration name is the current configuration name of the credstore; then returns nothing if they are different.

`xos_currentconfig`
Returns the configuration name of the credential assigned to the credstore, if it is not empty.

`xos_deletecred`
Deletes the current credential from the credstore, if any.

`xos_settimeout`
Sets a timeout (in seconds) after which the credential is purged from the credstore. Setting a new timeout always overwrites the previous timeout. Zero value cancels the timeout.

`xos_storecred [<filename>[<configname>]]`
If a filename is specified, this command saves in the credstore the contents of the file. If a configname is specified, it also fills in the current configuration

name with the configname. If no filename is specified, it will read the content from standard input.

This tool should not be used in a normal session. The PAM module in resource nodes and the `startxtreemos` tool in client nodes should be responsible for storing the credential in the credstore. This utility is useful just for testing purposes, or if machine administrators prefer to pre-load a credential manually.

`xos_credstored`

This daemon is automatically launched when needed, and so ordinary users may ignore this command. Advanced users may start it manually if preferred.

This executable is the daemon that implements the user-space credstore (`uskeystore`). The KRS implementation therefore does not use this daemon. A session credstore (which can be started with the command `startxtreemos -s`, see below) does not use this daemon either[11].

The daemon creates a Unix socket in the user's home directory, named "socket_xoscredstore_<uid>". Only programs with the same UID as `xos_credstored` can use the socket to communicate with the daemon.

`xos_credstored` daemons run until the machine is powered off, because they are shared among different sessions of the same user and the credentials survive the session that stored it. But the daemon will eventually die if the credentials time out or are deleted (with `delete_cred`).

On startup, `xos_credstored` tests that the same daemon is not already running. The test consists in a special request to the daemon using the aforementioned Unix socket.

2.6.2 Client node command line tools

`startxtreemos[-m] [-c <configuration_name>] [-t <seconds_timeout>] [-s] [<program_name> [<program_parameters>] | -]`

This multipurpose program is designed to run one or more of this tasks:

- Loading credentials in the current user credstore or starting a new session credstore and loading credentials in it.
- Running a program or a script read from standart input, which has access to the credstore.
- Setting a credential timeout.

The `-c` option is used to specify the configuration name of the requested credential to use. If no configuration_name is specified, it is assumed that the user would like to use the credential currently present in the credstore; if the credstore is empty, the "default" credential is requested.

[11] Session credstores are implemented as children of the process that launched the session, and they die when the parent process dies.

The `-t` option is used to set a timeout (expressed in seconds) over the credential. When the timeout expires, the credential is removed. A zero value cancels timeout.

The `-s` option launches a program (or a shell, if no program specified) in a private session, with its own credstore that is destroyed when program/shell ends.

A program to run may be specified, or use "-" parameter to get shell commands from `stdin`. If no program_name or "-" parameter is specified, then a shell is executed (if using "-s") and the `ENV` variable is defined with value `/etc/xos/shrc_xtreemos`. Otherwise, `startxtreemos` only guarantees that a credential is available in the credstore and sets the timeout, if specified.

`startxtreemos-ams [-c <configuration_name>] [-r] [<program_name> [<program_parameters>] | -]`

This program obtains the credential corresponding to configuration_name (or gets it from the current user credstore if available) and runs the program specified (or a shell if none is specified) with a new UID. This UID is computed using the Account Mapping System rules (see [11]) and the data present in the certificate of the credential.

The `-c` option is used to specify the configuration name of the requested credential to use. If no config_name is specified, it is assumed that the user would like to use the credential currently present in the credstore; if the credstore is empty, the "default" credential is requested.

The `-r` option replicates the credential in the credstore of the user that invokes `startxtreemos-ams`. The goal of this action is to cache the credential and avoid asking the user again if he prefers running other commands with `startxtreemos-ams` using the same configuration.

A program to run may be specified, or use "-" parameter to get shell commands from `stdin`. If no program_name or "-" parameter is specified, then a shell is executed and `/etc/xos/shrc_xtreemos` is specified in the `ENV` variable. Before the program/shell is executed, current environment is cleared and variables `HOME`, `SHELL`, `PATH`, `USER`, `LOGNAME` and `XOS_ENV` are filled with appropriate values.

`xos_getdumpcred [<configuration_name>]`

This program is a usage example of `libxos_getcred`. It dumps the current credential present in the credstore; if the credstore is empty, or a configuration_name is specified and the credential in the credstore is labeled with a different configuration name, `startxtreemos` is invoked to load the new credential in the credstore.

`xos_getdumpcred` reads the configuration file `/etc/xos/configname_alias` to convert the configuration name from any of its aliases, as defined in that file.

`read_protected_credfile <configuration name>`

Users are not expected to run this program directly. Instead, it is designed to

be invoked by `startxtreemos`. This program is a helper utility, *setgid* to group "startxtreemos", that reads a credential from `/etc/xos/creds` that ordinary users cannot read because of lack of permissions. This program is also used to ask the user for a password to decrypt the credential in case it is encrypted.

2.7 API

2.7.1 `libcredstore` API

This API is mainly of interest for programmers that would like to integrate `libcredstore` in a new project or extend it with a new module. The last updated API documentation is always included with the project itself, using the `doxygen` utility.

The usage of the API is very simple: a program just needs to include `credstore.h` in the code and invoke `get_creds_store_funcs()`. This function returns a `struct creds_store_funcs` with pointers to functions (e.g. `delete_cred` member is used to revoke a key, `store_cred` is used to save the key, `retrieve_cred` to get the key...). There also exists a `get_creds_store_funcs_by_impl`, in order to retrieve a specific implementation of the credstore. Current implementations include:

`krs` the usual Key Retention Service-based implementation.

`zkrs` is the same as krs, but in this case `store_cred` compresses the key and `retrieve_cred` uncompresses it. Compress ratio gain is variable, but as a reference, the size of `/etc/xos/creds/config2.pem` example credential is reduced to 69% of the original size.

`uskeystore` (user-space keytore) this implementation is based on a daemon per user (or per session). It is designed to make it easier to modify the implementation to use a system daemon, instead of a daemon per user, or to adapt the module to use e.g. the GNOME keyring[12].

Current implementation of `get_creds_store_funcs` returns zkrs pointers if KRS is available, otherwise returns `uskeystore` function pointers.

Caveats and known issues

To ease debugging, the Makefile compiles `libcredstore` with the `USE_CREDSTORE_IMPL_ENV` variable defined. With this compilation option, `get_creds_store_funcs` first tests if environment variable `CREDSTORE_IMPL` is defined; in this case, it invokes `get_creds_store_funcs_impl` with the variable value. However, it is safer

[12] The SVN version of gnome-keyring has exciting features like a PCKCS#11 module or integration with ssh-agent (see http://live.gnome.org/GnomeKeyring/Cryptoki)

for a production-strength library to be compiled without this option (leaving the variable undefined).

These are the function pointer members of `struct_creds_store_funcs`[13]:

`int store_cred(char *data_in_pem_format);`
This functions stores the credential that the user passes in PEM format (base64 encoded). If other credential is already stored, the new credential will overwrite it. Returns 0 on success, -1 on error.

`char* retrieve_cred (char *config_name);`
If `config_name` is NULL, this function returns the credential stored in the credstore (e.g. in Key Retention Service). If config_name is not NULL, first it checks if `config_name` is the current configuration name in the credstore; returns NULL if they are different.

`int check_cred_is_available (char *config_name);`
This function is similar to `retrieve_cred`, but returns 1 if the credential to this `config_name` is available, 0 otherwise (instead of the credential/NULL).

`int set_cred_timeout (int timeout);`
This function sets a timeout (in seconds) after which the credential is purged from the credstore. A zero value cancels the timeout. Returns 0 on success, -1 on error.

`int delete_cred ();`
Purges the credential from the credstore. Returns 0 on success, -1 on error.

`char * get_current_cred_configname ();`
This function returns the current configuration name of the key stored in the credstore. The initial value is "default". Returns NULL on error.

`int set_current_cred_configname (char *config_name);`
This functions sets the current configuration name of the key stored in the credstore. Returns 0 on success, -1 on error.

`char * join_new_session ();`
Up to now, all functions operate over the current user credstore. This function creates a new credstore session and all successive new calls of the process and its children will use this new credstore session. It is important in this case to call `exit_session` at the end of the session (although currently `krs` and

[13]This API is not considered stable yet, and could be changed in the future if more functionality is needed from it.

uskeystore do not need it because the session will be killed anyway when the program ends).

In the KRS implementation, this function invokes the syscall to start a new session. In the uskeystore implementation, this functions starts a new daemon using a fork and defines an environment variable with the address of the socket.

The function returns an environment value that the user must use with putenv, if launching other processes within the same session. The KRS implementation returns nothing, but uskeystore needs to modify the environment because new session processes use XOS_CREDSTORE_SOCKET to locate the socket of the session credstore daemon and to read the cookie and file handler required to authenticate.

```
int exit_session ();
```

If a user program calls join_new_session, it must call this function to end the session. In the KRS implementation this function does nothing (since the keystore session ends when the program that started the session ends). In the uskeystore implementation, this function kills the session credstore daemon, but this call is not needed in Linux because join_new_session calls prctl to establish that the credstore daemon receives a SIGTERM signal when the parent process dies. Returns 0 on success, -1 on error.

```
int join_default_user_session ();
```

If a program changes the UID, it must call this function to change to the new user credstore. In the KRS implementation, this function invokes the syscall to change the keystore. In the uskeystore implementation, it launches a new xos_credstored daemon. Returns 0 on success, -1 on error.

Caveats and known issues

If an application changes user, it must change the UID, not just the EUID, because in KRS the UID is used to select the user keyring[14]. With KRS, the FSUID (File system UID, which initially has the same value as the EUID) is used to assign the owner of a new created entry. If the program does not need to recover root privileges, the best solution is to change both the UID and the EUID. If the program needs to recover root privileges, the best option is continue with EUID=0, and change the UID with setreuid, and change the FSUID with setfsuid.

2.7.2 libxos_getcred API

This API just includes one function:

[14] With the uskeystore implementation, actually EUID is used to authenticate the socket instead of UID, but this is transparent to developers, because the module internally swaps UID and EUID before connecting with the daemon and restores them afterwards.

```
char * xos_getcred(char *configuration_name);
```

This function returns a credential in PEM format. The API uses `libcredstore` to implement single sign-on: if the credential is stored in the credstore, it is automatically retrieved, but if the credstore is empty or the credential is labelled with other configuration name different from the `configuration_name` parameter, `startxtreemos` (or `startxtreemos-m` depending on the compilation option which was set in the Makefile) is launched (with `configuration_name` as the paremeter, if not NULL) to obtain a credential that is then saved in the credstore.

The `configuration_name` parameter can be NULL. In this case, if the credstore is not empty, the credential is accepted without checking the configuration name registered. If the credstore is empty, a new credential is stored with configuration name "default".

To compile an application using this library, use the `-lxos_getcred` compilation option. The source code must include `xos_getcred.h`.

The `xos_getdumpcred.c` file is a minimalistic example application: it just obtains the credential and shows it on the screen.

2.8 Usage: testing that everything works correctly

2.8.1 `libcredstore`-related tools usage

The `libcredstore` package also includes two executables designed to test the implementation. They are located in `./test/test_credstore` and `./test/run_credstore_session`.

`test_credstore` This utility works in the following way:

First, it changes the configuration name to "configtest" and then retrieves the configuration name; it should show "configtest". Then, it stores "key value" and invokes `check_cred_is_available` and `retrieve_cred`, with "configtest" as the configuration name; it should show "true" and "key value". This test is then repeated, but using "fake" as the configuration name; it should show "false". And then, it does the same without any configuration name; it should show "true" and "key value". Afterwards, it deletes the credential and invokes `check_cred_is_available` and `retrieve_cred`; it should return "false". At last, it checks the timeout: first, it inserts a new value (remember that the credential was deleted in the last step) and invokes `set_cred_timeout` with a timeout of 4 seconds; then invokes `check_cred_is_available` and `retrieve_cred` after sleeping 2 seconds, and then after 2 additional seconds: it should show "true", "key value", "false".

run_credstore_session is a utility that launches a shell between a call to `join_new_session` and `exit_session`; that is, a new credstore is running until user runs "exit". With the `uskeystore` implementation, a `ps` will reveal two instances of `run_credstore_sesssion`: the second one is the credstore daemon.

You may check what implementation (`zkrs` or `uskeystore`) is being used in your system. For example, invoke `xos_storecred`, then type "hello" and press "Ctrl-D". If you are using `uskeystore`, then `ps -ax` will show a `xos_credstored` process which implements your credstore. If you are using zkrs and `keyutils` package is installed, `keyctl show` should display several entries, one of them starting with "x509uk_xos_u" and containing the credential, and another one named "xos_configname" and containing the configuration name corresponding to the credential. If you repeat this test after running `run_credstore_session` and your system uses `uskeystore`, `ps -ax` will not show the `xos_credstored` process, but `run_credstore_session` process will appear twice. The second process is the session credstore process.

2.8.2 startxtreemos-m, libxos_getcred

You may test `startxtreemos-m` and `libxos_getcred` by running `xos_getdumpcred`.

First, run `xos_dumpcred config1`. It will not show anything, because the credstore is empty. Now run `xos_getdumpcred config1`; this command will obtain the credential and show the content in the display. If we run `xos_getdumpcred` again, it will now display the credential. Run `xos_settimeout 5`; if you run inmediatly `xos_dumpcred`, the credential is shown, but after 5 seconds it will have been deleted. To obtain the credential again, run `xos_getdumpcred config1` or `startxtreemos-m -c config1` (the only difference is that `startxtreemos-m` does not show the credential, and allows to establish a timeout in the credential). If we change the configuration (e.g. run `xos_getdumpcred default`), then a new credential is loaded in the credstore that will replace the credential obtained with the "config1" configuration.

Results are more interesting if we compile the project with the `USE_READ_PROTECTED_HELPER=1` option. In this case, group owner of `/etc/xos/creds` is "startxtreemos" and only users in this group may access this directory.
`read_protected_credfile` may read the contents of this directory because it does not have the appropriate *setgid* attribute ("startxtreemos"). Invoking `xos_getdumpcred config1` implies that
`read_protected_credfile` is invoked and a dialog will ask for confirmation to the user for reading the file `/etc/xos/creds/config1.pem`. Subsequent calls to `xos_getdumpcred` show the credential without showing the dialog, because credential is retrieved from the credstore. But if we delete the credential with `xos_deletecred` (or use `xos_settimeout` to expire it), the

next time that we invoke
`xos_getdumpcred`, the dialog will ask for confirmation again.

If invoking `startxtreemos -c config` ends with the error message: "Sorry, you are not authorized to read credential file.", without displaying the dialog, this means that there is a error in the execution of
`read_protected_credfile`. You may try to invoke it manually with
`read_protected_credfile config1` to debug the problem.

Now we may test a credstore session, invoking
`startxtreemos -c config1 && startxtreemos -c config3`. The first command loads the credential referenced by configuration "config1" in the user credstore; the second command starts a shell and a session credstore, with the credential assigned to configuration name "config3". We can see the credential in the session credstore with `xos_dumpcred`, by running `xos_currentconfig` (it will display "config3") and finally deleting the credential with `xos_deletecred`. After we type `exit`, the session credstore is killed, and `xos_dumpcred` will show credential "config1".

2.8.3 `startxtreemos`, `startxtreemos-ams`

We may check `startxtreemos` with the same test applications as
`startxtreemos-m`. There are no differences in functionality, just in implementation: `startxtreemos` uses the PAM module while `startxtreemos-m` doesn't. If we receive an error such as "Error checking user authentication: Permission denied", that means that there is some problem in the PAM module installation; please revise the corresponding installation instructions. One visible difference between
`startxtreemos` and `startxtreemos-m` is that the former previously verifies the certificate, while `startxtreemos-m` just interprets the credential as a plain text.

Finally, we may also test `startxtreemos-ams`. First, ensure that you previously started the `xos_amsd` daemon as root. Run `startxtreemos-ams -c config1`; you will obtain a shell with a different UID; you may check this with the `id` command. If `libcredstore` is using the `uskeystore` implementation, a `xos_credstored` daemon will be running with that UID, and this daemon will continue to run after the user has closed the session. If we prefer that this daemon does not continue running, invoke `xos_deletecred` or use `xos_settimeout`, because the daemon dies when there is no credential to store anymore, and it is autolaunched when the function to store a credential is invoked (provided that the daemon is not already running).

Chapter 3

Terminal Mobility

Terminal mobility modules in XtreemOS-MD allow users to stay connected to the grid while they are moving, maintaining the same IP address and with only minimum delays when handing off between access networks. As it is explained in previous deliverables [13], this mobility can be attained by the usage of Mobile IPv6 protocols [7]. The software used in XtreemOS-MD for this purpose is an adaptation for ARM architectures of the USAGI MIPv6 implementation [10], which has been incorporated to the latest mainline Linux kernels.

3.1 System requirements

3.1.1 Hardware

Since XtreemOS-MD is designed at this stage for PDAs, it is mandatory to have an ARM architecture PDA, equipped with a WiFi network interface.

As of this writing, tests have only been conducted on **HP Ipaq hx4700** PDAs and QEMU emulators.

3.1.2 Software

The terminal mobility modules consist of a `kernel driver` and a **daemon** running on user space. The kernel driver installation requieres to have Linux kernel compiled enabling the following options:

- CONFIG_EXPERIMENTAL=y
- CONFIG_SYSVIPC=y
- CONFIG_PROC_FS=y
- CONFIG_NET=y
- CONFIG_INET=y

- CONFIG_IPV6=y
- CONFIG_IPV6_MIP6=y
- CONFIG_XFRM=y
- CONFIG_XFRM_USER=y
- CONFIG_XFRM_SUB_POLICY=y
- CONFIG_INET6_XFRM_MODE_ROUTEOPTIMIZATION=y
- CONFIG_IPV6_TUNNEL=y
- CONFIG_IPV6_ADVANCED_ROUTER=y
- CONFIG_IPV6_MULTIPLE_TABLES=y
- CONFIG_INET6_ESP=y
- CONFIG_NET_KEY=y
- CONFIG_NET_KEY_MIGRATE=y

These options are needed just to communicate with any other MIPv6 node. If we want our node to act as a Mobile Node (which we certainly will), more options must be enabled too:

- CONFIG_IPV6_SUBTREES=y
- CONFIG_ARPD=y

These mobility kernel extensions are available starting from kernel version 2.6.20. In case of having an older version, kernel source must be patched in order to get these capabilities.

3.2 Installation manual

The basic XtreemOS-MD version, targeted at PDAs, is based on the Ångström distribution. To enable the aforementioned kernel extensions, it must be recompiled and normally a new flashable operating system image must be generated. Ångström images are generated using OpenEmbedded [9], a development environment that allows users to target a wide variety of embedded devices, including Linux PDAs and mobile phones. Thus, the steps to get one of these images with a recompiled kernel are the following:

3.2.1 OpenEmbedded installation

In order to get OpenEmbedded installed, please follow the instructions in the Ångström website (see [2])

3.2.2 Machine and distro selection in OpenEmbedded

Since XtreemOS-MD is still in early development stages, desktop emulation is the best way to test it, for example using the QEMU emulator, a multi-architecture emulator available in a number of diferent platforms, including Linux.

As OpenEmbedded is able to generate targets for multiple devices, it can be specified QEMU as the target machine when building the operating system image, and to specify Ångström as the Linux distribution to generate.

In order to set this options, the developer must edit `local.conf` file, which is located in `/OE/build/conf/` folder. The developer has to add/modify the `MACHINE` value to "`qemuarm`" and the `DISTRO` value to "`angstrom-2007.1`". In case we are targeting a real device, like the Ipaq hx4700, `hx4700` is the correct value for `MACHINE`.

It is also mandatory, if we plan to use QEMU as the testing platform, to add the option for ext2 filesystem images, by adding "`ext2`" to `IMAGE_FSTYPES` (space separated).

3.2.3 Kernel modification

Kernel configuration files for OpenEmbedded are located in

```
org.openembedded.angstrom-2007.12-stable/packages/linux
```

Each target machine has assigned a specific kernel whose configuration is in a subdirectory under this folder. For QEMU images, the assigned kernel is `linux-rp` and there are several versions the user can choose from.

Each version folder has got a number of `defconfig` files, one for each machine that uses this kernel. Modification of this file is the way to enable or disable kernel extensions when OpenEmbedded builds it.

For example, in order to get the mobility extensions enabled on a QEMU Ångström version, using kernel version 2.6.23, the following file must be modified:

```
/OE/org.openembedded.angstrom-2007.12-stable/packages/linux/linux-rp-2.6.23/defconfig-qemuarm
```

Modify the lines as outlined above (see section 3.1.2), or add them if they're not listed.

3.2.4 OS image generation

After the modification of the kernel configuration file, the OS image must be generated.

```
# set environment variables
source source-me.txt
#Go to the OE tree
cd /path/to/org.openembedded.stable
#Make sure it is up to date
mtn pull ; mtn update
#Start building
bitbake base-image ; bitbake console-image ; bitbake x11-image
```

The generated image will be located under

`build/tmp/angstrom/deploy/glibc/images/qemuarm`

This image can later be used to launch Ångström under QEMU, for example, or to be flashed onto a real PDA.

3.2.5 User tools installation

Once we have the kernel mobility extensions enabled on the kernel and the MIPv6 tools have been built, we have to copy the `mip6d` executable (the Mobile IPv6 daemon) to the `/usr/bin` folder, and give it execution permissions.

3.3 Configuration

The MIPv6 configuration file is located in `/usr/local/etc`. In case it is missing, it must be created, otherwise default options will be applied.

In this file, all configuration lines must be terminated with a semicolon. Subsections are enclosed in brackets ("{" and "}") and string values will be quoted with double quotes ("").

The following options are available:

3.3.1 Common options

NodeConfig <CN | HA | MN>; Indicates if the daemon should run in Correspondent Node, Home Agent or Mobile Node modes.

Default: CN

DebugLevel <number>; Indicates the debug level of the daemon. If the value is greater than zero, the daemon will not detach from tty (i.e. debug messages will be printed on the controlling tty).

Default: 0

DoRouteOptimizationCN <boolean>; Indicates if a node should participate in route optimization with a Mobile Node.

Default: enabled

NonVolatileBindingCache <boolean>; This option is currently ignored. Binding cache is always stored in volatile memory, and is not retained between shutdown and startup

3.3.2 Options common to Home Agents and Mobile Nodes

Interface name;

Interface name { MnIfPreference <number>; IfType <CN | HA | MN>; } Specifies an interface and the options associated with it. If no options are present, Interface can be terminated with a semi-colon. This is used for home agents to specify which interfaces are used for HA operation. For the home agent to work properly, a Router Advertisement daemon (e.g. `radvd`) must broadcast advertisements with the Home Agent bit and Home Agent Information option set on those interfaces. This option is also used by multihomed Mobile Nodes to define which interfaces are used by it.

`MnIfPreference` sets the interface preference value for an interface in a multi-homed Mobile Node. The most preferred intefaces have preference 1, the second most preferred have 2, etc. A preference of zero means the interface will not be used.

Default: 5

`IfType` overrides the default node behavior for this interface. If a MN doesn't wish to use this inteface for mobility, or a node doesn't act as HA on this interface, the interface type should be set to CN.

Default: same as NodeConfig

UseMnHaIPsec <boolean>; Indicates if the MN-HA MIPv6 signalling should be protected with IPsec.

Default: enabled

KeyMngMobCapability <boolean>; If dynamic keying with MIPv6-aware IKE is used, this options should be enabled. It turns on the K-bit for binding updates and binding acknowledgements.

Default: disabled

IPsecPolicySet { HomeAgentAddress address; HomeAddress address/length; IPsecPolicy ... ... } `IPsecPolicySet` is a set of policies to apply for matching packets. A policy set can contain multiple HomeAddress options, but only one HomeAgentAddress option. Under Home Agent configuration, the HomeAgentAddress field contains its own address, and home address fields may contain any number of mobile nodes for which the same policy applies.

`IPsecPolicy` has the following format:

`IPsecPolicy type UseESPnumber <number>;`

Field `type` can be one of "HomeRegBinding", "Mh", "MobPfxDisc", "ICMP", "any", "TunnelMh", "TunnelHomeTesting", or "TunnelPayload". The "any" option protects all transport layer communication between the MN and HA. Currently, only the ESP IPsec protocol is supported, but in the future AH and IPComp might also be available. The two remaining numeric fields are the IPsec reqid values, the first one used for MN - HA, the second one for HA - MN communication. If just one value is defined, the same reqid will be used in both directions. If no reqid is given, reqid will not be used.

If more than one IPsec transport mode or tunnel mode policy is defined between the MN and HA in each direction, reqid can be used to provide an unambiguous one-to-one mapping between IPsec policies and SAs. Otherwise the policies will just share a common SA.

3.3.3 Home Agent-specific option

HaMaxBindingLife <number>; Limits the maximum lifetime (in seconds) for Mobile Node home registrations.
Default: 262140

SendMobPfxAdvs <boolean>; Controls whether home agent sends Mobile Prefix Advertisements to mobile nodes in foreign networks.

SendUnsolMobPfxAdvs <boolean>; Controls whether home agent send unsolicited Mobile Prefix Advertisements to mobile nodes in foreign networks.

MinMobPfxAdvInterval <number>; Sets a minimum interval (in seconds) for Mobile Prefix Advertisements.
Default: 600

MaxMobPfxAdvInterval <number>; Sets a maximum interval (in seconds) for Mobile Prefix Advertisements.
Default: 86400

BindingAclPolicy <address> <allow | deny> Defines if a MN is allowed to register with the HA or not. The MN home address of the MN is given in the address field.

DefaultBindingAclPolicy <allow | deny> Defines the default policy if no matching BindingAclPolicy entry is found for a MN.
Default: allow

3.3.4 Mobile Node-specific options

MnMaxHaBindingLife <number>; Limits the maximum lifetime (in seconds) for Mobile Node home registrations.
Default: 262140

MnMaxCnBindingLife <number>; Limits the maximum lifetime (in seconds) for Mobile Node Correspondent Node registrations.
Default: 420

MnDiscardHaParamProb <boolean>; Toggles if the Mobile Node should discard ICMPv6 Parameter Problem messages from its Home Agent. As the ICMPv6 error messages will not normally be protected by IPsec, a malicious third party can quite easily impersonate the HA to the MN. Once the MN has accepted these messages, Denial of Service attacks are possible, even though its home registration signalling is protected by IPsec.

Default: disabled

SendMobPfxSols <boolean>; Controls whether mobile node sends Mobile Prefix Solicitations to the home network.

DoRouteOptimizationMN <boolean>; Indicates if the Mobile Node should initialize route optimization with Correspondent Nodes.

Default: enabled

MnUseAllInterfaces <enabled | disabled> Indicates if all interfaces should be used for mobility. The preference of these interfaces is always 1. Unless you use dynamically created and named network interfaces you should normally disable this option and use Interface options to explicitly list the used interfaces.

Default: disabled

UseCnBuAck <boolean>; Indicates if the Acknowledge bit should be set in Binding Updates sent to Corresponent Nodes.

Default: disabled

MnRouterProbes <number>; Indicates how many times the MN should send Neighbor Unreachability Detection probes to its old router after receiving a Router Advertisement from a new one. If the option is set to zero, the MN will move to the new router straight away.

Default: 0

MnRouterProbeTimeout <decimal>; Indicates how long (in seconds) the MN should wait for a reply during an access router Neighbor Unreachability Detection probe. If set, it overrides any default Neighbor Solicitation Retransmit Timer value greater than MnRouterProbe Timeout. For example, if the interface Retransmit Timer is 1 second, but MnRouterProbeTimeout is just 0.2 seconds, the MN will only wait 0.2 seconds for a Neighbor Advertisement before proceeding with the handoff.

Default: 0

OptimisticHandoff <enabled | disabled> When a Mobile Node sends a Binding Update to the Home Agent, no Route Optimized or reverse tunneled traffic is sent until a Binding Acknowledgement is received. When enabled, this option allows the Mobile Node to assume that the binding was successful right after the BU has been sent, and does not wait for a positive acknowledgement before using RO or reverse tunneling.

Default: disabled;

MnHomeLink <name> { HomeAddress <address/length>; HomeAgentAddress <address>; MnRoPolicy ... ... } Each MnHomeLink definition has a name. This is the name (enclosed in double quotes) of the interface used for connecting to the physical home link. To set up multiple Home Addresses on the Mobile Node, you need to define multiple MnHomeLink structures. The interface names don't have to be unique in these definitions. All the home link specific definitions are detailed below:

HomeAddress <address/length>; Address is an IPv6 address, and length is the prefix length of the address, usually 64. This option must be included in a home link definition.

HomeAgentAddress <address>; Address is the IPv6 address of the Mobile Node's Home Agent.

Default: ::

The route optimization policies are of the form:

MnRoPolicy <address> <boolean>; Any number of these policies may be defined. If no policies are defined default behavior depends on the DoRouteOptimizationMN option.

The fields for a route optimization policy entry are as follows: address defines the Correspondent Node this policy applies to; if left undefined, the uspecified address is used as a wildcard. The boolean sets route optimization either on or off for packets matching this entry.

3.4 Command line tools

MIP6D

Although not really intended for command-line usage by humans, the MIPv6 daemon has the following syntax:

```
mip6d -c <file> -d <level> [-C|-H|-M]
```

Where:
`-c <file>` reads configuration from this file
`-d <level>` sets the debug level
`-C o -correspondent-node` The node is a Correspondent Node (CN)
`-H o -home-agent` The node is a Home Agent (HA)
`-M o -mobile-node` The node is a Mobile Node (MN)

3.5 Usage

Next, a sample configuration file (mip6d.conf) will be shown for both, Home Agent and Mobile Node.

Home Agent

```
NodeConfig HA;

Interface "eth0";
  Interface "eth1";

  UseMnHaIPsec enabled;

  IPsecPolicySet {
   HomeAgentAddress 3ffe:2620:6:1::1;

      HomeAddress 3ffe:2620:6:1::1234/64;
      HomeAddress 3ffe:2620:6:1::1235/64;

      IPsecPolicy HomeRegBinding UseESP;
      IPsecPolicy TunnelMh UseESP;
}
```

Mobile Node

```
NodeConfig MN;

DoRouteOptimizationCN enabled;

  DoRouteOptimizationMN enabled;

  UseCnBuAck enabled;

  MnHomeLink "eth0" {
HomeAgentAddress 3ffe:2620:6:1::1;
      HomeAddress 3ffe:2620:6:1::1234/64;

      #           address             opt.
      #MnRoPolicy 3ffe:2060:6:1::3    enabled;
      #MnRoPolicy                     disabled;
  }

  UseMnHaIPsec enabled;

  IPsecPolicySet {
   HomeAgentAddress 3ffe:2620:6:1::1;
      HomeAddress 3ffe:2620:6:1::1234/64;
```

```
      IPsecPolicy HomeRegBinding UseESP;
      IPsecPolicy TunnelMh UseESP;
}
```

Chapter 4

Future work

All the features, processes and information contained in this document applies to the XtreemOS-F software packages for mobile devices as they are released as of this writing (May 2008). However, the development of XtreemOS-MD is an ongoing work, which will progress in the near future along two main lines of development:

- An evolutive line of development, which includes support and bugfixes for the current release, and the improvement of current methods implemented in this first basic version of XtreemOS-MD. These improvements will include:
  - Design and development of new utilities for managing and configuring user profiles (with regard to authentication methods for accessing the grid), in order to make the access to the grid easier and more transparent.
  - Development of a library in order to integrate XtreemOS authentication methods with existing remote shell applications like SSH/Telnet without having to modify their code.
  - Modification of `libxos_getcred` to integrate it transparently with any user application, without needing to modify its code.
- Another line directed to the development of the advanced version of XtreemOS-MD, intended to be run in mobile phones. This line comprises several objectives:
  - Porting of all the existent software in the basic version of XtreemOS-MD to the more restricted environment of a mobile phone, including future bugfixes and new features described above.
  - Integration of VO support mechanisms with Java applications.
  - Evaluation of the usage of virtualization technologies to provide enhanced isolation between grid and mobile phone features, so that critical phone functionality cannot be affected by grid processes.

References

[1] The Ångström Distribution.
http://www.angstrom-distribution.org.

[2] Ångström Development page.
http://www.angstrom-distribution.org/building-angstrom.

[3] Ångström Package Repository.
http://www.angstrom-distribution.org/repo/.

[4] Fabrice Bellard. QEMU documentation.
http://fabrice.bellard.free.fr/qemu/qemu-doc.html.

[5] Official GridLab web site.
http://www.gridlab.org/.

[6] Tao Guan, Ed Zaluska, and David De Roure. A grid service infrastructure for mobile devices. In *Proceedings of the First International Conference on Semantics, Knowledge and Grid*, page 42, 2005.

[7] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775 (Proposed Standard), June 2004.

[8] University of Louisiana. Official WSRF.NET web site.
http://www.cs.virginia.edu/~gsw2c/wsrf.net.html.

[9] OpenEmbedded web site.
http://www.openembedded.org/.

[10] USAGI Project Homepage.
http://www.linux-ipv6.org/.

[11] XtreemOS Consortium. Design and Implementation of Node-level VO Support D2.1.2. Integrated Project, December 2007.

[12] XtreemOS Consortium. Design of a Basic Linux Version for Mobile Devices D2.3.3. Integrated Project, December 2007.

[13] XtreemOS Consortium. Requirements and Specifications of a Basic Linux Version for Mobile Devices D2.3.2. Integrated Project, June 2007.

Appendix A

Development environment for XtreemOS-MD

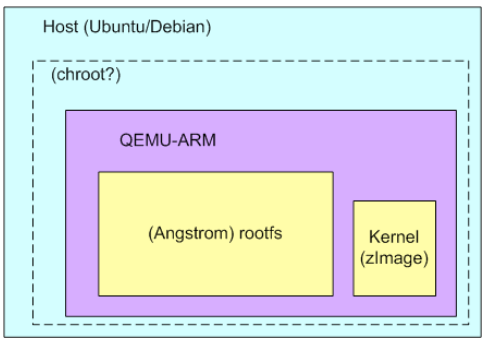
In order to ease the development of mobile/embedded applications in XtreemOS-MD (and of XtreemOS-MD components themselves), which is known to be costly in terms of time and developer effort and learning curve, a special development environment has been devised.

Here, a brief description of its current state and usage is given. However, it is still an ongoing work, and more features will be added to it as the project evolves.

A.1 The environment

Currently, the develpment environment for XtreemOS-MD consists of an ARM architecture emulator (namely, QEMU-ARM), which the developer uses to edit, build and execute the source code, as if working on a real PDA. This emulated machine works mainly with two disk partitions (stored in files in the host machine), one for the kernel and another one for the root filesystem. This latter one contains the files from an Ångström Linux distro (the distro which XteeemOS-MD for PDAs is based upon).

A rough schema of the environment is showed in the following figure:



The usage of virtual environments like VMWare to follow this manual is discouraged in terms of speed, as one would be developing in an emulator inside another emulator. Other kinds of environments like chroot could be used, since they do not put such a penalty on efficiency, and could serve to isolate this environment from the host.

A.2 Generating a development environment

Although already-made development environments are normally available from the SVN repository of the project (both for GUI interfaces and for console-mode interfaces), here we will describe how to generate one such environment.

In order to generate a XtreemOS-MD machine fit for development, a special image must be generated, with much more root filesystem space to let development tools work with freedom, and to store object files and other by-products of typical development. XtreemOS-MD user images usually have a size of 64 or 128 MB (the same amount that a typical device normally has for flash ROM). In this case, as XtreemOS-MD development will be done in a QEMU environment, more space is available from the host system and thus, developers can make use of it.

The process for generating such an image is the following:

A.2.1 Generating a bigger image

Modify the file

```
/OE/org.openembedded.angstrom-2007.12-stable/conf/machine/include/qemu.inc
```

in your OpenEmbedded installation and change the value of `IMAGE_ROOTFS_SIZE_ext2` to the desired size in bytes. It's recommended to use a size above 512 MB (524288).

A.2.2 Regenerating the OS image

Following the installation instructions shown in the Terminal Mobility chapter (chapter 3), we must use OpenEmbedded to get a new image with the new size, launching the image generation process as explained there.

Once the image is generated, it can be launched with the QEMU emulator, as described on its user manual [4]. For development images, it is recommended to use emulator memory values of 248MB and above.

A.2.3 Connecting to XtreemOS-MD via SSH

By default, XtreemOS-MD has a SSH server running on startup, so it will probably be easier to connect to the virtual machine through SSH from now on (specially if we plan to make extensive use of command line tools). Of course, network must

be set up adequately to redirect the emulator's SSH port to a determined port in the host system.

Then, we can initiate a SSH session from the host or any other machine:

```
ssh [-l <user>] [-p <port>] <IP-angstrom>
su
(insert XtreemOS root password)
ipkg update
```

This last command updates the software package list with the latest information from the online repositories. From now on, if the network connection is configured correctly, we will have at our disposal a wide variety of packages, that can be installed through the `ipkg` utility, in a similar way as `apt-get` works in Debian/Ubuntu.

A.2.4 Installing the compiler tools

To install the development packages, type in your emulator:

```
ipkg install binutils gcc libc6-dev gcc-symlinks \
 binutils-symlinks autoconf automake perl-module-file-path \
 make coreutils
```

If you wish to know which other packages are available, you can query Ångström's package repository [3].

Also, you can search for packages from inside Ångström with `ipkg`:

```
ipkg list
ipkg list | grep <string>
```

or, to see the installed packages:

```
ipkg list installed
```

In case the DNS of the emulated machine gets unconfigured, we will have to edit `/etc/resolv.conf` manually and add the following:

```
nameserver <DNS>
```