INGI113 **Minix3 Project** 2013-2014
Adding a transparent file encryption support to the Minix File System

Michael Saint-Guillain and Laurent Lesage

November 5, 2013

Contents

1 Introduction

The opportunity of protecting the data stored on a file system is a valuable property for an operating system. With a regular file system, anyone who has a physical access to a block device can easily extract the data; therefore using encryption methods in the file system management part of an operating system is an efficient way to guarantee that the data are well protected, but without obligation for the end user to systematically perform encryption tasks.

2 Minix 3

Minix 3 is a POSIX compatible micro-kernel OS. Micro-kernel approach is based on the ideas of modularity and fault confinement.

"*MINIX 3 is a free, open-source, operating system designed to be highly reliable, flexible, and secure. It is based on a tiny micro-kernel running in kernel mode with the rest of the operating system running as a collection of isolated, protected, processes in user mode.*" [minix3.org]

2.1 Reference book

We strongly recommend for this project that you obtain a copy of Andrew Tanenbaum "*Operating Systems Design and Implementation* (third edition)", a.k.a. "*The Minix book*".

Performing this project definitely involves that you deeply understand the Minix architecture; you are therefore expected to read the sections below **before performing any modifications in the Minix code**. However, you must be aware that the following list is *not exhaustive* (but could also be too much complete ..!) :

- Section 1.3: Concepts
- Section 1.4: System calls
- Section 3.4: Overview of I/O in Minix 3
- Section 3.5: Block devices in Minix 3
- Section 5.3: File system implementation
- Section 5.6: Overview of the Minix 3 file system
- Section 5.7: Implementation of the Minix 3 file system

Due to the amount of material you will have to read, we can't but advise you to start reading as soon as possible.

2.2 Virtual File System (VFS)

Sections 5.6 and 5.7 tackle the standard Minix 3 file system, MFS. However, since version 3.1.3 Minix actually implements a Virtual File System (VFS) server. This server is an abstraction layer to the actual different file system servers, offering an unique interface to such that applications can request the various file system types (like MFS) in a uniform way. The good news is that this interface makes it easier to introduce new file systems. Documentation about the VFS-FS protocol can be found at

http://wiki.minix3.org/en/DevelopersGuide/VfsFsProtocol

Documentation about VFS internals is also available:

http://wiki.minix3.org/en/DevelopersGuide/VfsInternals

Furthermore, documentation about Minix Virtual File System design and implementation issues is available in B. Gerofi's (A. Tanenbaum's student) master thesis, which is fortunately provided by the Minix community:

www.minix3.org/doc/gerofi_thesis.pdf

2.3 How to use Minix 3

Minix is available on computers in *Intel* room, via `Qemu` virtualization tool. To make your job easier, you are provided a `Makefile` that handles some recurrent operations. Remember that you can always work remotely (e.g. from your own place) to computers in *Intel* room via `studssh` gateway:

```
$ ssh login@studssh.info.ucl.ac.be
login@studssh.info.ucl.ac.be's password:
...
-bash-3.2$ ssh login@machine.info.ucl.ac.be
login@machine.info.ucl.ac.be's password:
...
-bash-4.1$
```

, where of course you have to replace `login` by your login and `machine` by the name of the computer in Intel room.

2.3.1 Initialization of your working directory

An image of a virtual disk containing a regular Minix operating system, `minix_orig.img`, is stored in `/etinfo/applications/minix`; in order to save place on your network directory, you will be working on a local virtual disk `minix_local.cow` which is a *Copy-on-Write* version of `minix_orig.img` .
`/dev/c0d1`
Fortunately, the `Makefile` sets up your working directory for you. All you need to do is to download the `Makefile` and place it in a newly created directory (you can call it `minix`) in your network home directory; then simply execute `make init` .

Your `minix` directory should afterward contain the following files and subdirectories:

minix_local.cow is your local *Copy-on-Write* version of `minix_orig.img` .

additional_disk.img is as its name suggests an additional disk image, useful for testing your implementation; it will be available in Minix under `/dev/c0d1` .

additional_disk_ext2.img is an additional disk formatted in *ext2* file system; available under /dev/c0d2.

src contains the Minix source code that you will be modifying.

src_orig contains the Minix original source code, for recovery purposes.

test the files placed in the `test` directory will appear into the `test` directory in your Minix home directory.

2.3.2 Running Minix

In order to run the virtual machine containing Minix, simply type:

```
-bash-4.1$ make run
...
Hit a key as follows:
   1  Start Minix 3
   3  Start Custom Minix 3
1
```

If you are running Minix for the first time, select option 1. Then login as root (no password needed) and configure the virtual machine so that it will be able to communicate with the host machine (you just have to perform this once):

```
# echo "export HOST_USERNAME=<votre_login_en_salle>" >> .profile
# echo "export HOST_MINIXPATH=<chemin_vers_projet>" >> .profile
# . .profile
```

where an example of `chemin_vers_projet` could be `/etinfo/users/2011/duchnock/minix` . You can now update both the Minix sources and the `test` directory in the virtual machine:

```
# ./update_minix
```

This script makes use of `rsync` over `ssh` to synchronize the two directories with the host machine. Once the virtual machine is up to date, you can compile the custom Minix system:

```
# cd /usr/src/tools
# make libraries hdboot
```

You can now reboot the virtual machine and boot on your custom Minix system:

```
# shutdown
...
Minix will now be shut down ...
d0p0s0> menu
Hit a key as follows:
   1  Start Minix 3
   3  Start Custom Minix 3
3
```

The virtual machine therefore boots on your new kernel. In order to simply shutdown the machine when you are done with it, type `off` instead of `menu`.

2.3.3 The Makefile is your friend

Typing make alone will show you what the Makefile can do for you:

```
-bash-4.1$ make
Utlisez 'make <target>' où <target> est :
  init     pour initialiser le répertoire du projet
  run      pour executer la machine virtuelle (en console)
  run_x11  pour executer la machine virtuelle (en fenêtre)
  patch    pour générer le patch
  dist     pour générer une archive comprenant le patch,
           le rapport et le dossier test
  clean    supprime l'archive et le patch
  mrproper supprime les disques virtuels
```

For instance, if you want to regenerate the virtual disk `additional_disk.img` , you just need to delete it and then run `make init` ; the same applies to the other files/directories originally generated by `make init` .

3 Objectives

In this section we will present you the different objectives of this project.

3.1 Introductory project

As a first visit through the maze of Minix source code we ask you to implement the following -quite simple- system call:

```
struct pid_s {
        pid_t me;
        pid_t parent;
};

/**
 * Out: the pids structure contains the PID (processus ID)
 *      of the calling process and the PID of parent process
 * Return: OK if success, otherwise returns a negative value
 *         and errno is set to the error code
**/
int getpidinfo(struct pid_s *pids);
```

The main objective is to identify the files to modify so that this new system call will be added to Minix [1]. A good start to understand how a system call is implemented in Minix is to trace its execution and identify what's going on. For instance, try to observe what happens between the call of `read` by a user-space process and its return.

Unlike past years, this introductory pre-project will be evaluated during the mid-term interview (see 4). Furthermore, presence of `getpidinfo` system call in your final product is mandatory.

3.2 Main project

Your ultimate goal is to implement a transparent file encryption support for the Minix File System (MFS). File encryption support means that Minix should be able to read and write encrypted files stored on an MFS file system; i.e. only the content of the files present on the file system will be encrypted. Note that your goal *is not about implementing a new file system support in Minix, but only a new file system access mode* for MFS.

Transparent means that once the (pseudo-)encrypted file system has been mounted with file encryption support, encryption operations must be totally transparent to user-space processes and file I/O libraries; for instance, you should hereafter be able to display the content of a file `foe.txt` in clear by using the regular command "`cat foe.txt`", whilst executing the same `cat` command on the file system but mounted without file encryption support would display encrypted contents. Similarly, writing into a file (e.g. using `echo`) should leave data encrypted afterwards if the FS was mounted with file encryption support.

[1] You are therefore **not** supposed to simply decompose this system call into two existing system calls that actually do jointly the same work, namely `getpid` and `getppid`.

3.2.1 Expected program

Your are expected to implement a special version of the `mount` minix command which takes a supplementary option `-e`, causing the target MFS file system to be mounted with file encryption support. Running this command in the Minix shell should give something like:

```
# cc -o test/mount test/mount.c
# mkdir /mnt/disk
# test/mount -e /dev/c0d1 /mnt/disk
Encryption key:
...
/dev/c0d1 is read-write mounted on/mnt/disk
#
```

3.2.2 Encryption

You are expected to protect the file contents by using a *cipher* encryption method. To this aim, you will make use of the `BF_cbc_encrypt` function provided by the OPENSSL library present on Minix. This function permits to encrypt (and decrypt) blocks of 8 bytes, using the *Bluefish* cipher encryption algorithm. For more information, read the corresponding man page.

However, instead of directly use an encryption method working with blocks of 8 bytes, we recommend you to start with implementing a trivial per-byte encryption method, and then go ahead with `BF_cbc_encrypt` once you are convinced that everything works as it should work. An example of a trivial encryption method would be to add to each byte a constant value, which depends on the encryption key.

3.2.3 Suggestions and hints

- Late night coding has always introduced even more bugs; plan your work and trust me, do not start this project three weeks ahead.
- Perform incremental changes: divide your problem into a sequence of easier sub-problems (divide & conquer !).
- Which parts of the Minix sources will you modify ? Which servers (VFS ? MFS ? Both ?)
- Discuss and justify your implementation and architecture choices (in the report).
- Write parts of the report before coding. For each relevant piece of code that you modify, it is usually better to first discuss it in your report; it will force you to really think at what your implementation goals and, in so doing, avoid inappropriate code. Furthermore, by writing the report later you risk to only document the few last details you tackled, or even fill your report with irrelevant contents.
- Add useful comments to your code.
- ...

3.3 Test strategies

For this project we do not expect you to provide unit tests for each function. We ask you to provide some script or code that will be placed in `test` directory and demonstrate that your system calls work as they should do. That directory must contain a `Makefile` with at least one target, *tests*, executing all the tests.

3.4 Final report

Your final report will be of maximum 8 pages [2] and be of *IEEEtran 10pt 1 column* format [3]. It must at least contain the following informations:

- Description of your architecture
- Description of the modified or added source files
- Encryption method used
- Achievements and further work
- Authors and group number !
- References
- ...

3.5 Deliverable

Your entire work, including the report and the implementation, must be delivered on Foditic, in section "Travaux". The `Makefile` is able to directly generate the archive that you will upload on Foditic. In order to do so, you must have placed your final report in the project directory and renamed it `rapport.pdf` . Then generate the archive by typing `make dist` .

4 Deadlines summary

There are roughly two deadlines for this project. For the *mid-term interview* you just need to bring yourself and be able to clearly explain us how you managed to implement `getpidinfo` syscall; also you must already have ideas in mind on how you plan to implement your file encryption support. The second deadline is about your final product. The deadlines will therefore be as follows:

- November 5th: Project documents available on Foditic
- November 28th and 29th: Mid-term interviews (about 10' per group, during TP session time slots)
- December 27th 23:00: projects delivery (on Foditic)

Regular TP sessions will still be organized between the deadlines, Tuesday 16:15 at Siemens room and Friday 10:45 at Intel room. Your presence to those sessions is **not** mandatory; however you should view those sessions as an opportunity to do planned group meetings and to directly ask (precise) questions to the assistants.

[2] This is an upper bound; if you think that you have enough with half of it then go with it, but please do not fill in your report with non-relevant information.

[3] You can rely on the following resources:

Latex: "howto" available within Document -> "Projet_Minix" section on Foditic

Word: template available at www.ieee.org/documents/trans_jour.docx

5 Evaluation

Your work will be (fairly) evaluated the basis of the following criteria:

- Mid-term interview (correctness and clarity of your presentation of the `getpidinfo` syscall, relevance of your future plans about the main project).
- Architecture of the system calls and new file system server implementation.
- Quality, exhaustive and relevance of the final report document.
- Compliance with Minix conventions.
- Quality of the code produced.
- Test results

Bonus points

Substantial bonus points can be granted if you manage to deal with some or all of the following advanced problems.

Multiple file system support

The minimum requirement is that your file encryption support applies to MFS file systems; a better implementation would generalize to any file system supported by the VFS. Groups that managed to solve this problem must adapt their test strategy in order to demonstrate the correctness of their approach.

Marking a (MFS) file system as being encrypted

A major issue regarding our (file-)encryption support framework is definitely that there is no way to know whether a given file system is actually (file-)encrypted. Therefore, whenever an (file-)encrypted (MFS) file system is mounted, one can always write information in it no matter if the encryption mode is activated (i.e. is mounted with file encryption support) or not. A way better approach would be to find a way to mark an encrypted file system such that if mounting it normally (i.e. without encryption support) would result in a read-only mounted file system. This problem involves, amongst others, the following question: how can you mark a Minix file system as being encrypted ? Groups that managed to solve this problem must adapt their test strategy in order to demonstrate the correctness of their approach.

Good work !