University of Liège

Operating Systems

Project 6 : Hidden files in vFat

3rd engineering bachelor

Authors :
Antoine Louis
Tom Crasset

Professor :
L. Mathy

Assistant :
T. Barbette

Academic Year 2017-2018

1 Introduction

The aim of the project was to hide or unhide given files. To do that, we must have unlocked first the file system with a password. So the usual steps to hide/unhide a file are the following ones :

1. Set a password to the file system (no password by default).
2. Unlock the file system with that password or none if no password is set (the file system is locked by default).
3. Hide/unhide a file.
4. Lock the file system.

2 Implementation

First of all, we added an attribute ATTR_PROTECTED in the `msdos_fs.h` file, which has the value 64 (01000000 in binary). This attribute is used to specify that a certain inode is protected or unprotected (hidden or unhidden is a synonym but there's already an attribute that hides a file in linux), if the attribute is set or unset respectively.

We also had to modify the ATTR_UNUSED value to include our attribute. To implement the different functions, we first thought to use system calls. We rapidly encountered a problem with the *kernpath()* function that would give us a dentry based on a user defined pathname, even after using *copy_from_user()*. Finally, we decided to solely use ioctl functions calls. We chose to add a few cases to the function *fat_generic_ioctl()*, which is called by *ioctl()* using flags, and thus we needed to define a few flags in `msdos_fs.h`. We just took the values following the already implemented flags. A small note is that during the installation of the kernel, one has to use the command 'make headers_install' , else this specific header (and others) won't get installed and thus only the old one would remain.

1. #define FAT_IOCTL_SET_PROTECTED _IOW('r', 0x14, __u32)
2. #define FAT_IOCTL_SET_UNPROTECTED _IOW('r', 0x15, __u32)
3. #define FAT_IOCTL_SET_LOCK _IOW('r', 0x16, __u32)
4. #define FAT_IOCTL_SET_UNLOCK _IOW('r', 0x17, __u32)
5. #define FAT_IOCTL_SET_PASSWORD _IOW('r', 0x18, __u32)

Once these constants defined, we implemented the different functions in `./fs/fat/file.c`. Most of the following functions were done in mutual exclusion using mutexes.

2.1 __fat_readdir()

This function is already implemented in `/fs/fat/dir.c`. It gets all directory entries from an inode struct and loops over and reads from them. By setting an if statement checking if the file system is locked and the file is hidden, we can bypass the reading and just go to the next entry.

2.2 fat_ioctl_set_protected()

Using a file pointer, we gain access to the inode and further to the super_block info and we can thus modify the attributes using *fat_save_attributes()*. We only add the attribute ATTR_PROTECTED and thus the protected bit is set to 1.

2.3 fat_ioctl_set_unprotected()

The implementation of this function is almost identical to *fat_ioctl_set_protected()* except that it does the opposite, namely setting the protected bit to 0.

2.4 fat_file_open()

This function overrides the *open()* function that is common to all file systems, but for FAT file systems. It prevents the operating system to open the file if it is hidden, which is not checked with the ioctl's because to use ioctl we first need to open the files.

2.5 fat_ioctl_set_password()

In the fat_boot_sector struct, there is an unused variable named 'hidden' of 32 bits. We use this variable to hold the 4 digit password that will lock or unlock our file system.

The function *fat_ioctl_set_password()* takes as arguments an inode structure and an attribute of 32 bits. The former permits to access the super_block and from there we can access the fat_boot_sector structure where the hidden variable is defined. The latter is the concatenation of the two passwords given by the user. As we can only pass a 32-bit value in this function, this value must hold both the current password and the new one. Thus, we include both by assigning the first, shifting 16 bits in either direction and finally doing an OR operation with the second. To get the two passwords out, we do the reverse. The figure 1 shows its representation.

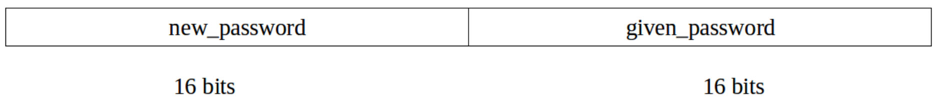

FIGURE 1 – Representation of the u32 argument.

2.6 fat_ioctl_set_unlock()

In the msdos_sb_info from the `fat.h` header, which holds all the information of a superblock, we added an int called 'unlock' that is either 1 or 0 and represents the state of the file system, either unlocked or locked respectively. This int will be zero by default, which means that our file system will be locked by default.

The *fat_ioctl_set_unlock()* function takes two arguments : the inode structure that will permit to access the fat_boot_sector structure in order to get the current set password, and an u32 attribute containing the password given by the user. In this function, we check that the given password corresponds to the current set one and, if it's the case, we unlock the file system by setting the 'unlock' variable of the msdos_sb_info structure to 0.

2.7 fat_ioctl_set_lock()

This function is quite simple given that we only need to access the 'unlock' variable and set it to 1. It thus only need the inode struct as argument.

3 Conclusion

Thanks to these implementations and manipulations, we managed to hide or unhide a given file. Notice that, for an unknown reason, the change takes some time to appear. So if we hide a given file after having unlocked the file system, the file will still be visible for approximatively 15 to 20 seconds and then will be hidden.