International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR)
ISSN 2249-6831
Vol. 3, Issue 1, Mar 2013, 341-350
© TJPRC Pvt. Ltd.

TRANS STELLAR
•Journal Publications •Research Consultancy

FILEPRO: FILE PROTECTION ON FILE SERVERS USING LINUX KERNEL MODULE

WAIEL M. YOUSSIF[1] & NOURELDIEN A. NOURELDIEN[2]

[1]Student, Deptatrment of Computer Science, University of Science and Technology, Omdurman, Sudan

[2]Associate Professor, Department of Computer Science, University of Science and Technology, Omdurman, Sudan

ABSTRACT

Most operating systems have protection mechanisms that prevent unauthorized disclosure and enforce integrity at a file level. Linux operating system and its variants contain several native protection mechanisms that enforce file access policy. These controls include file access permissions, users' grouping and access control lists.

In many situations, we may need to allow a user to prohibit his file(s) from all system users including super users (not root user) and some users within the file(s) group, while allowing specific group user(s) to access the file(s). Such a situation can not be achieved by using the existing file access controls provided in Linux operating systems such as ACL.

In this paper, we provide a file access mechanism called "FILEPRO", a loadable Kernel module that allows users to protect their own files from super users and some group users, while granting an access level to certain group members. In Linux-based systems, Kernel source consists of modules that are enhanced to release new versions of the operating systems. FILEPRO can be one of those modules, which can be deployed into new Kernel's releases.

KEYWORDS: File Protection, Loadable Kernel Module, Access Control List, Super Users

INRODUCTION

Linux-based operating systems are well secured in terms of file permissions and access control. However, they suffer from one fundamental flaw which is the root user privilages. Root privileges are the powers that the root account has on the system. The root account is the most privileged account on the system and has a complete access to all files and commands. Also root's powers includes the ability to modify the system in any way desired and to grant the ability to read, modify and execute specific files and directorie) for other users, including any of those that are by default reserved for root [12].

This unlimited power of root user leaks a security flaw if an intruder grants a root access. Therefore, in real implementation of servers, root user is not used for system login. Moreover, some organizations, from security point of view, restrict the root access and provide system administrators with different super users accounts to operate their tasks. Super users unlike the root user are not granted unlimited priveliges. Also there is a possibility to create different super users, with various user identifiers, which will allows monitoring of super users actions. If all administrators use the root user, then it becomes impossible to differentiate who did what.

In Linux operating system, users can create files and set their required permissions. File permissions are set on three levels, which are the owner, the owner's group and other users. In a shared networking environment, like a file server or a web server, users may need to create their own data files with a higher level of security that allows certain group members to read the files, and prevent other group members from reading or copying the files.

Linux systems provides ACL as a mechanism to deal with shared files, but it lacks many features such as protecting files from super users, support of all file-systems types (For instance, in Kernel version 2.6.9 ACL does not support the Reiser file-system), needs a file-system to be remounted with the ACL option and it is editable only by the root.

In this paper we design and develop a loadable kernel module named FILEPRO which overcomes ACL limitations. FILEPRO can protect the file from super users, and it is essentially designed to use a generalized method of files' protection that is supported under all file-systems types, it does not necessitate a remount of the file-system and with FILEPRO only the file owner has the control over his protected files and users' authorization..

Since files are the core of information in all operating systems, the FILEPRO file protection mechanism is also applicable in other different areas apart from file servers, such as database files, application server files, web servers, etc.

The rest of this paper is organized as follows; in section 2, the required background topics are covered, in section 3 the basic design features of FILEPRO are discussed, section 4 shows testing and results and finally conclusions are drawn in section 5.

BACKGROUND

On Linux-based systems, everything is a file; if something is not a file, it is a process. Programs, services, texts, images, and so forth, are all files. Input and output devices, and generally all devices, are considered to be files. However most of files in Linux-based systems are considered to be regular files. Text files and executables are of the regular files type. Other file types are classified into directories, special files, links, sockets, and named pipes [5].

i Node Structure

One of the main features of file-systems is the i-nodes (index nodes) of every file, which can be defined as a data structure on a file-system that stores all the information about a file except its name and its actual data. Each inode describes a data structure on the hard disk, storing the properties of a file, including the physical location of the file data. At the time a new file is created, it gets a free inode.

System Calls

A system call is the invocation of an operating system routine. Operating systems contain sets of routines for performing various low-level operations. If a certain application needs to execute an operating system routine from a program, it has to make a system call.

In computing, a system call, or software interrupt is the mechanism used by an application program to request service from the operating system. System calls often use a special machine code instructions which causes the processor to change mode (e.g. to "supervisor mode" or "protected mode"). This allows the OS to perform restricted actions such as accessing hardware devices or the memory management unit. Modern processors can typically execute instructions in several, very different, privileged states. In systems with two levels, they are usually called user mode and supervisor mode.

As in figure (1), when a system call is invoked, the invoking program is interrupted, and information needed to continue its execution later was saved. The processor then begins executing the higher privileged code, which, by examining processor state determines what is being requested.

A Kernel's address list, called the system call table, contains the addresses of system calls routines. The Kernel starts looking for the address of the requested system call to be executed. When it is finished, it returns to the program, restoring the saved state, and the program continues executing [7].

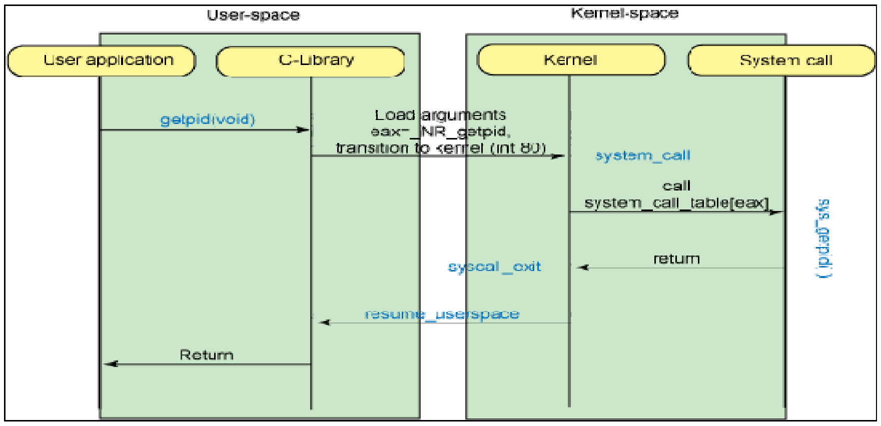


Figure 1: Invoking a System Call [7]

Users' Classification and Permissions

Users in Linux operating systems can be classified into different ways. If we think about the users' powers and privileges then we classify system users in the following categories

- Root user with unlimited privileges on all system resources. The root user can modify, remove or create any file (file or directory), add system users, run or kill any process. The root user can also switch to any other user without a need for a password.
- Super users with unlimited privileges on specific system resources. A super user can have root privileges on certain areas. The super user account can be configured to have all the root privileges with restriction on certain resources.
- Regular users with limited privileges on system resources. They have complete control on their home directories, but they can run certain shell commands based on their granted permissions.

The root user does not require a permission to access a file or modify its contents. Super users can run root commands that are entitled to [8].

Kernel Modules

Kernel Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system [1]. For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system. Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image. Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we add new functionality. When an application program issues a system call, Linux transfers the execution from user (lowest) level to the Kernel (highest) level. A Kernel code, demonstrated by a module, handles the system call and performs the required operation. The Kernel will be able to access the user address space to deliver the required output to the application process.

Unlike users' programs, Kernel modules run in the Kernel address space to be linked to the Kernel and use its headers and structures.The compiled binary object of a program can be found in the form *name.o*, where the 'o' extension stands for object. The Kernel module object is extended by ko, e.g. *kernel_module.ko*, where 'ko' stands for Kernel object. When a system program or a device driver is built into a Kernel object, the Kernel should know about it. In other words, the Kernel object needs to be loaded and linked to the Kernel. In Linux, the Kernel module object can be loaded by using the *insmod* command [3].

When the module is loaded, the Kernel needs to find an initialization function, generally named as init_module, to load and register the module into the Kernel address space. If the initialization is successful, the module can invoke Kernel procedures and functions which do not involve the C library. For example, the module can use the Kernel printk function which behaves similarly to the standard C function printf. The Kernel does not use the printf and it has its own defined printing function, because it should run independently without the use of C library [1].

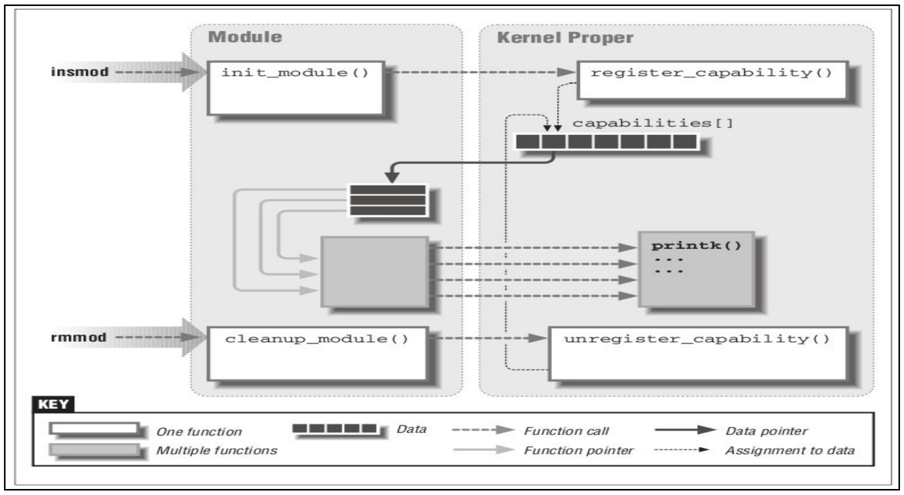


Figure 2: Loading & Unloading Kernel Modules [1]

Figure (2) depicts the module initialization. It also shows how a module can be unloaded from the Kernel address space. The *rmmod* command can be invoked to unload a module.The Kernel needs to find the cleanup_module procedure. The procedure is invoked to perform the termination pre-requisites, such as the deletion of a special file that had been created by the module. When the module is unloaded, it is out of Kernel address space, and it no longer exists in the memory.

Communication to Kernel Modules

The /proc file system is used to exchange information with processes. A Kernel module and processes can communicate through the /proc file system. The Kernel module can have a file entry in the /proc file-system which is used for communication from and to a user-level program. The file is a symbolic object for a memory address location of the Kernel module. When a user program needs to send information to a Kernel module, it writes to the /proc file entry. If a user program needs to receive information from a Kernel module, it reads the /proc file entry [2].

FILEPRO DESIGN

FILEPRO was designed to have certain advantages over normal Linux security mechanisms, specially ACL. FILEPRO can protect the file from super users, and it is essentially designed to use a generalized method of files' protection that is supported under all file-systems types, it does not necessitate a remount of the file-system and with FILEPRO only the file owner has the control over his protected files and users' authorization. In this section we show how FILEPRO is designed to achieve the above security requirements.

FILEPRO Components

The FILEPRO consists of two main components, namely FILEPRO Kernel Module and i-PRO a User Interface Program. The FILEPRO Kernel module has to prevent copy or access to protected files from unauthorized users, deny the creation of hard links for protected files and disallow any other Kernel module to take control from the FILEPRO, or bypass its file access protection.

The i-PRO User Program should provide communication to the FILEPRO Kernel module, specify files to be protected, add or deny users who can access the protected files and list users' protected files.

FILEPRO Kernel Module Design

The design of the FILEPRO Kernel Module depends four mechanisms, system call interception, linked list structures anda hash function.

System Calls Interception

FILEPRO can read the address of the open system call and store it in a variable. Then, as in figure (3), the Kernel module modify the system call table by altering the address of the open system call to point to FILEPRO open-routine.

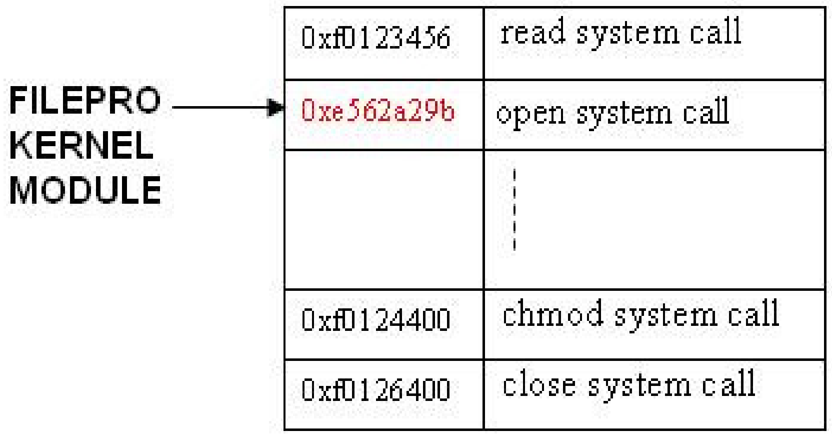


Figure 3: Modifying the System Call Table

Thus when the Kernel handles users open requests, it locates and calls the filepro_open_routine instead. In the filepro_open_routine, the module may deny all the open requests or accept them based on certain criteria.

Linked Lists Structures

FILEPRO maintains a list of protected files and their authorized users. The lists of protected files and authorized users compose what is so-called the linked lists structure. To keep track of protected files and authorized users FilePro maintains two lists; one for the protected files and the other for authorized users.

However, there are two conditions that arise out here, the first is that, the list of protected files needs to be endless since restricting the number of protected files is not acceptable in a file server scenario, and the second is that both lists need to be inter-related, so FILEPRO can know which users are authorized to access which file.

It is obvious that linked lists has to be used to satisfy the two conditions, Therefore FILEPRO maintain two linked lists structures. The first list contains the file hash value which is a numeric representation of a file name, the number of authorized users, a pointer to the list that contains the identifiers of authorized users, ad a pointer to next protected file.

The second list is an extended list of each node in the first list and it contains the identifiers of authorized users and a pointer to next authorized user of the file. Hence, the final design of the linked lists structure was realized by figure (4).

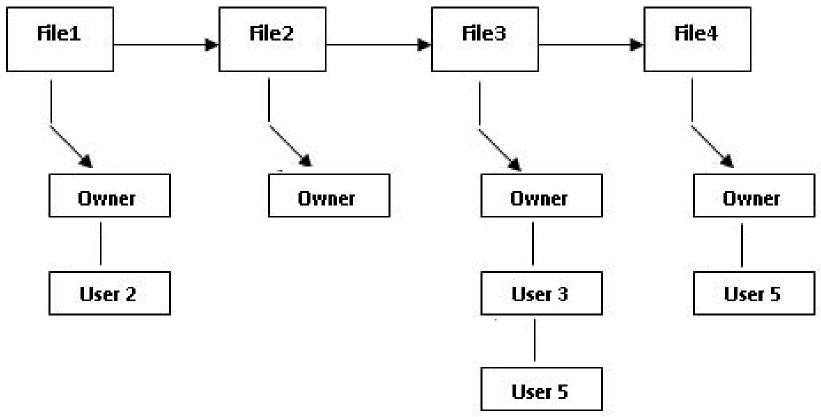


Figure 4: The Design of Linked Lists Structure

The Hash Function

To secue protected files, FILEPRO keeps a list of hashes to identify the protected files. When a file is accessed, it calculates its hash value and compares it with the ones on the list. If they match, then the file is a protected one. FILEPRO uses the ELF hashing algorithm. The ELF hashing is considered to be one of the best algorithms that have a minimum possibility of collisions [10].

i-PRO Design

i-PRO is a user program which communicates to the FILEPRO so that users can specify which files they need to protect. Hence, a two-ways communication will be required. i-PRO is used to inform FILEPRO which files to protect and which users to authorized. Consequently, FILEPRO needs to feedback i-PRO whether the requested operation was succeeded or not.

The communication between the user program, i-PRO, and the Kernel module, FILEPRO, is held through a /proc file-system entry. This latter is created by FILEPRO in such a way that normal users will not be able to read or write to it. The effective user ID [11] of the i-PRO program belongs to the root user, and hence the program will be able to read or write to the /proc file entry regardless of the user who runs it.

When i-PRO needs FILEPRO to protect a file, it sends three parameters, the owner of the file, the file hash value and a user to be authorized for file access. When FILEPRO receives the i-PRO message, it adds the necessary information to the linked lists and feeds back the i-PRO. The message sent from i-PRO to the FILEPRO is encoded in a proprietary format.

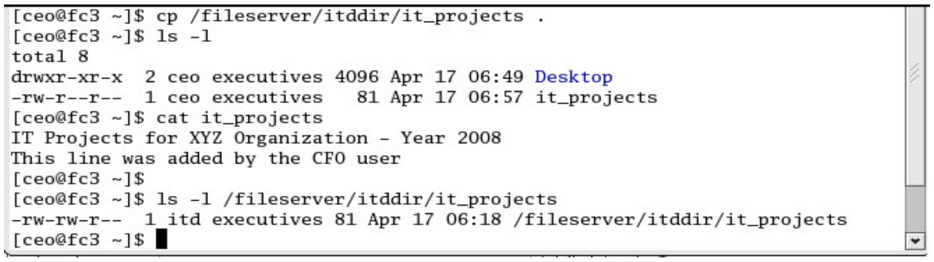
TESTING AND RESULTS

To test the FILEPRO we assume that we have the following five users that belong to one group called “executives”; CEO: The user of chief executive officer, CFO: The user of chief financial officer, ITD: The user of IT director, HRD: The user of human resources director and CTO: The user of chief technical officer.

The five users can create files in the file server, and assign read permissions for their group, which allows files sharing between them. We assume that the ITD user needs to create a new information file, regular data file, for IT objectives and projects. He wants to allow the CEO and CFO users to access the file, while prohibiting the HRD and CTO users from accessing the file. Sub-grouping of the users into new different groups will not resolve the problem since the sharing of some other data files must be preserved.

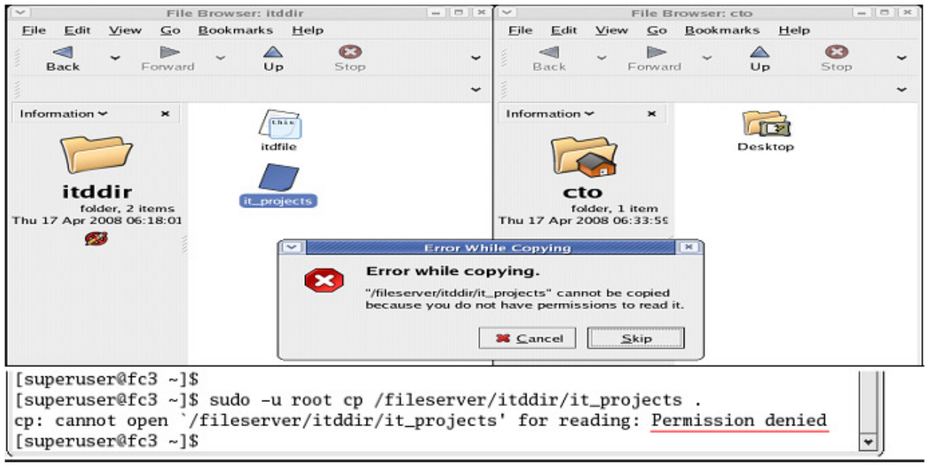
The testing of FILEPRO will comprise of the following: copying a protected file by authorized and unauthorized users, opening a protected file by authorized and unauthorized users, creating a hard link of a file by authorized and unauthorized users and append the file by authorized and unauthorized users.

Figure (5), demonstrates that the CEO will be able to copy the protected file to his home directory while figure (6) shows that the super user and the CTO were prevented from doing so.



```
[ceo@fc3 ~]$ cp /fileserver/itddir/it_projects .
[ceo@fc3 ~]$ ls -l
total 8
drwxr-xr-x  2 ceo executives 4096 Apr 17 06:49 Desktop
-rw-r--r--  1 ceo executives   81 Apr 17 06:57 it_projects
[ceo@fc3 ~]$ cat it_projects
IT Projects for XYZ Organization - Year 2008
This line was added by the CFO user
[ceo@fc3 ~]$
[ceo@fc3 ~]$ ls -l /fileserver/itddir/it_projects
-rw-rw-r--  1 itd executives 81 Apr 17 06:18 /fileserver/itddir/it_projects
[ceo@fc3 ~]$
```

Figure 5: CEO User Copies the IT Projects File

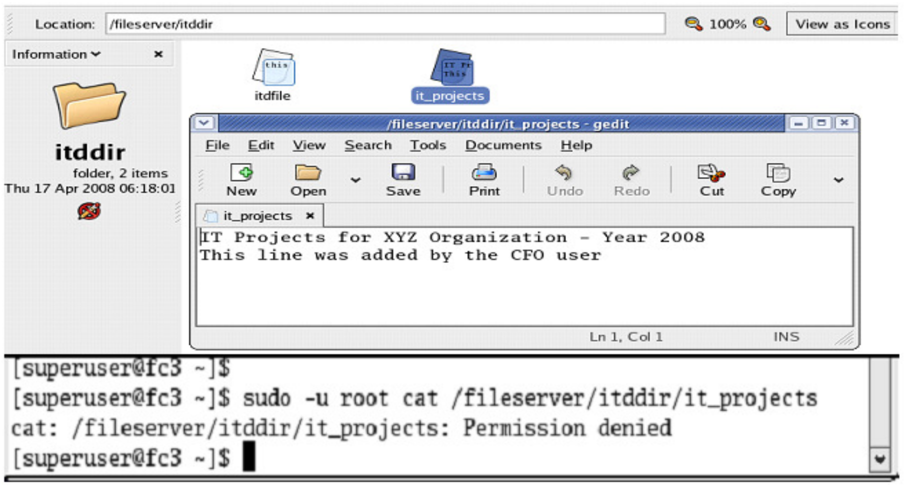


Error while copying.
"/fileserver/itddir/it_projects" cannot be copied because you do not have permissions to read it.

```
[superuser@fc3 ~]$
[superuser@fc3 ~]$ sudo -u root cp /fileserver/itddir/it_projects .
cp: cannot open `/fileserver/itddir/it_projects' for reading: Permission denied
[superuser@fc3 ~]$
```

Figure 6: CTO and Super Users Revoked from Copying the IT Projects File

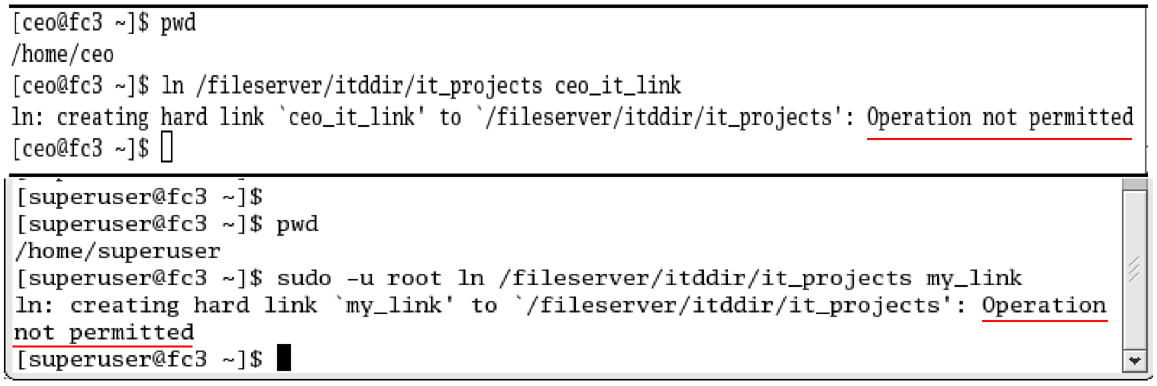
Figure (7), shows that the CFO will be able to use the gedit text editor to view the IT Projects files while the super user was prohibited from viewing it.


Location: /fileserver/itddir
100%
View as Icons
Information
itdfile
it_projects
itddir
folder, 2 items
Thu 17 Apr 2008 06:18:01
/fileserver/itddir/it_projects - gedit
File Edit View Search Tools Documents Help
New Open Save Print Undo Redo Cut Copy
it_projects
Ln 1, Col 1
INS

```
IT Projects for XYZ Organization - Year 2008
This line was added by the CFO user
```

```
[superuser@fc3 ~]$
[superuser@fc3 ~]$ sudo -u root cat /fileserver/itddir/it_projects
cat: /fileserver/itddir/it_projects: Permission denied
[superuser@fc3 ~]$
```

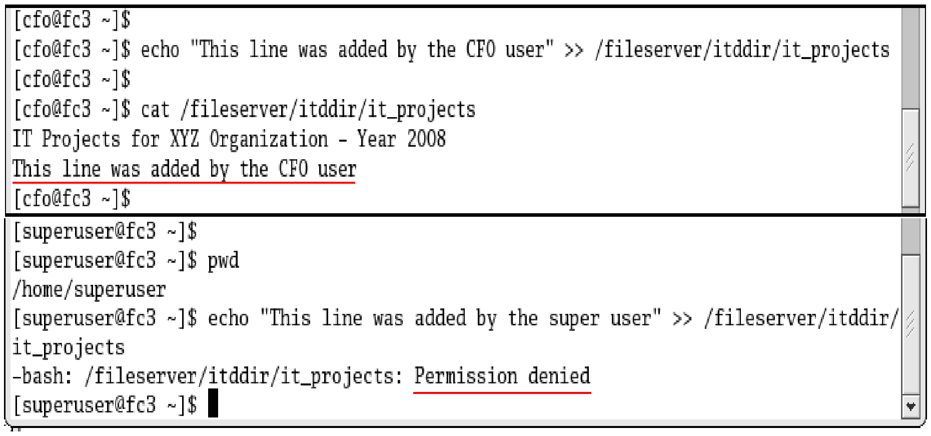
Figure 7: CFO User Viewing the IT Projects File



```
[ceo@fc3 ~]$ pwd
/home/ceo
[ceo@fc3 ~]$ ln /fileserver/itddir/it_projects ceo_it_link
ln: creating hard link `ceo_it_link' to `/fileserver/itddir/it_projects': Operation not permitted
[ceo@fc3 ~]$
```

```
[superuser@fc3 ~]$
[superuser@fc3 ~]$ pwd
/home/superuser
[superuser@fc3 ~]$ sudo -u root ln /fileserver/itddir/it_projects my_link
ln: creating hard link `my_link' to `/fileserver/itddir/it_projects': Operation
not permitted
[superuser@fc3 ~]$
```

Figure 8: CEO and Super Users Revoked from Creating Hard Links of the IT Projects File



```
[cfo@fc3 ~]$
[cfo@fc3 ~]$ echo "This line was added by the CFO user" >> /fileserver/itddir/it_projects
[cfo@fc3 ~]$
[cfo@fc3 ~]$ cat /fileserver/itddir/it_projects
IT Projects for XYZ Organization - Year 2008
This line was added by the CFO user
[cfo@fc3 ~]$
```

```
[superuser@fc3 ~]$
[superuser@fc3 ~]$ pwd
/home/superuser
[superuser@fc3 ~]$ echo "This line was added by the super user" >> /fileserver/itddir/
it_projects
-bash: /fileserver/itddir/it_projects: Permission denied
[superuser@fc3 ~]$
```

Figure 9: Response to CFO and Super Users when Appending to the IT Projects File

CONCLUSIONS

File protection is a major security issue that must be handled by operating systems, specially in networking environments where users can share their files and a non-trivial file protection requirement may arise. FILEPRO is one possible solution for these requirements, and a command line programs, such as i-PRO, are more efficient in making communication to Kernel modules because they can be used by system users who connect remotely, through TELNET, to their servers.

FILEPRO was designed to provide a reliable level of security for regular files, such as data files in a file server, database server or web servers.

REFERENCES

1. Peter Jay Salzman & Michael Burian, "The Linux Kernel Module Programming Guide", Version 2.6.3, 2005.
2. Alessandro Rubini & Jonathan Corbet, "Lniux Device Drivers", O'REILLY Associates, 2nd Edition, 2001.
3. James Mohr, "The Linux Tutorial", http://www.linux-tutorial.info/modules.php?name=Tutorial&pageid=5, visited on 19/Mar/2012.
4. Anand Santhanam, "Towards Linux 2.6, "http://www.ibm.com/developerworks/linux/library/l-inside.html, visited on 19/May/2007.

5. Machtelt Garrels, “General Overview of the Linux File System”,
http://www.faqs.org/docs/linux_intro/sect_03_01.html, visited on 31/Jan/2012.

6. Andries Brouwer, “Memory”, http://www.win.tue.nl/~aeb/linux/lk/lk-9.html, visited on 20/March/2012.

7. Brian Smith, “UNIX System Calls”, http://www128.ibm.com/developerworks/linux/library/l-system-calls/, visited on 02/Feb/2012.

8. “LinuxUsersandSudo”,http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch09_:_Linux_Users_and_Sudo, visited on 10/Feb/2012.

9. Jenkins Bob, “Hash Function”, http://en.wikipedia.org/wiki/Hash_function, visited on 19/Feb/2012.

10. JulienneWalker,“TheArt of Hashing”, http://www.eternallyconfuzzled.com/tuts/algorithms/jsw_tut_hashing.aspx, visited on 20/Feb/2012.

11. DeanMah,“LinuxFilePermissionsandSetuid,Part2”,http://www.evolt.org/article/UNIX_File_Permissions_and_Setuid_Part_2/18/263/, visited on 15/March/2012.

12. "What is root? - definition by The Linux Information Project http://www.linfo.org/root.html retrieved 28/Feb/2012