ISSN No. 0976-5697

Volume 4, No. 3, March 2013 (Special Issue)

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Security Systems: A concept through Encrypting File System

Shruti Jain
M.Tech Scholar, Singhania University,
Junjhunu, Rajasthan
jainshruti1988@gmail.com[1]

Chintal Kumar Patel
M.Tech Scholar, Singhania University,
Junjhunu, Rajasthan
smilingchintal@gmail.com[2]

Abstract: The need for data security emerges from the widespread deployment of shared file systems, greater mobility of computers and the rapid miniaturization of storage devices. It is increasingly obvious that the value of data is much more than the value of underlying devices. The theft of a personal laptop or a thumb drive leaves the victim vulnerable to the risk of identity of theft in addition to the loss of personal or financial data and intellectual property. An encrypting file system employs secure and efficient mechanisms to encrypt or decrypt data on-the-fly as it is being written to or read from the underlying disk, to provide a level of data privacy that goes beyond simple access control. Also, issues, such as trust models, backups and data recovery must be resolved. An encrypting file system must also be tightly integrated with the operating system for ease of use and flexibility. Although the design of such system is a well-researched problem, existing implementations still lack the security and usability features that must be present in a truly scalable system that can be successfully deployed in enterprises. This paper includes study of existing system. It also represents the conceptualization, design and implementation of a kernel-space encrypting file system that incorporates an advance key management scheme to provide a high grade of security while remaining transparency and usability.

Keywords: Encrypted File System, Virtual File System (VFS), Access Control List (ACL), File System Key (FSK).

I. INTRODUCTION

Security of the stored data on disk is largely neglected area, the theft of the stored data may cause loss of personal information. The theft of stored data can be done through coping data from the system via any thumb devices. To ensure security from such kind of theft, the obvious solution through restricting users to use any thumb device especially pen drives. But such kind of restriction causes many problems because now a day use of thumb devices is a must for working properly, there is a huge amount of data transfer regularly on such device. This paper is based on the study for encrypting a file system so that without restricting users, purpose of data security must be ensured. For such purpose Linux operating system's file system is the selected file system.

The Paper starts with the Scope of the research, then it depicts the introduction of the Linux file system, leading forward towards the design issues in converting a file system into an encrypted one, then at the end it represent the proposed implementation of such encrypted file system.

II. SCOPE OF ENCRYPTING A FILE SYSTEM

Although the design of the encrypting file system is a well - researched problem, all existing solution still lack the necessary security and usability features that are desired from a truly enterprise ready system. The first and foremost solution for resolving the theft problem via thumb devices, is to block access from such device so we have to disable all the USB devices. In Linux this task will be done through following procedures:

Process 1: **Step1**: Login as root.
Step 2: Open /etc/modprobe.d/no-usb for edit. (create this file if not exist)
Step 3: Append this line to the no-usb file and save it.
install usb - storage /bin/true
Step 4: Reboot server to enforce changes just made.
Alternative process
Process 2: Append "blacklist usb_storage" to the module
vim /etc/modprobe.d/blacklist

Through these procedures we can disable all the USB devices (Pendrives, USB harddisk etc) so that no-one copies the data from the system disk. But this solution is not feasible because now a days, the use of USB devices becomes so common that restricting USB devices leads us into a trouble pool.

By introducing a Encrypting File System the whole purpose of securing the data from the theft is being done in a secure and transparent way. The Encrypting File System, encrypts and decrypts the data on-the-fly i.e. whenever the data is being read from the disk, it encrypts the data through the encrypting key and whenever this data is being written on any other disk via any USB device, is being decrypt by the file system automatically. Therefore in any case, if the designated file system doesn't have the decrypting key the data will be useless for data system thus protected from the theft. This whole task is being done in such a transparent way, that if the thief doesn't knows about the encryption and decryption key, the data he had being theft is of no use and even he doesn't recognizes about the changes in file done through the encryption.

III. ARCHITECTURE OF ENCRYPTED FILE SYSTEM (LINUX)

The Architecture of the kernel is shown in the following figure:

CONFERENCE PAPER
II International Conference on
"Advance Computing and Creating Entrepreneurs (ACCE2013)"
On 19-20 Feb 2013
Organized by
2nd SIG-WNs, Div IV & Udaipur Chapter , CSI , IEEE Computer Society Chapter India Council ,
IEEE Student Chapter Geetanjali Institute of Technical Studies, Udaipur, Rajasthan, India

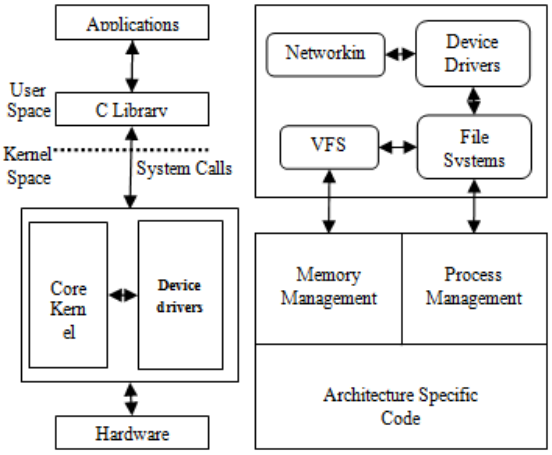

Figure 1 Architecture of Linux Kernel

Linux is a monolithic kernel. Device drivers and kernel extensions run in kernel space, with full access to the hardware, although some exceptions run in user space. Unlike standard monolithic kernels, device drivers are easily configured as modules, and loaded or unloaded while running the system. Also unlike standard monolithic kernels, device drivers can be pre-empted under certain conditions. This latter feature was added to handle hardware interrupts correctly, and to improve support for symmetric multiprocessing.

To convert such Kernel into a kernel that support the encrypted file system, the changes made through introducing an encrypted core instead of the core kernel. we describe the software modules that need to be implemented in the Linux kernel and allied user-space support utilities. We need to change the core with the encrypted core, also we need a super block to perform the basic encryption and decryption operations. We have introduced Crypto API and PKI support for the storage of public as well as private keys, and we required the authentication so we stored user certificates and CA certificates. The Changes are illustrated in the following figure:

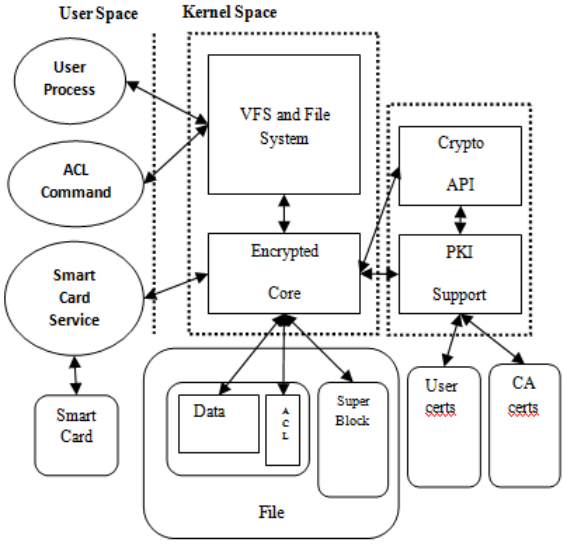

Figure 2 Architecture of Encrypted Kernel

IV. CRYPTOGRAPHIC FILE SYSTEM: DESIGN AND IMPLEMENTATION

A. Preconditions:

The following pre-requisite activities must first be carried out when cryptographic file system is being deployed in an enterprise environment:

a. A public and private key pair must be generated for all users in the authentication domain who require access to encrypted file systems.
b. The public keys must be signed by an appropriate CA and the certificates made publicly accessible in the certificate repository. The trusted certificates store must be established for all computer systems that mount encrypted volumes. Smart cards, if used, must be issued to users.
c. The data recovery agent account must be established in the authentication domain and its certificate added to the repository. The corresponding private key may be split into multiple smart cards and issued to different persons.

B. Encrypted File System Creation:

The on-disk superblock structure of the underlying file system and the corresponding mkfs command are suitably modified to take the following actions when creating an encrypted volume:

a. The block cipher algorithm to be used for symmetric key encryption, chaining mode, key size (of FEKs and the file system key), IV generation method, message authentication code algorithm and the user's choice regarding page cache encryption are specified on the mkfs command line. These parameters are appropriately encoded and stored in the superblock.
b. The file system key FSK is randomly generated and encrypted using key material derived from a passphrase (or an external trusted hardware to avoid trusting the administrator). The result is stored in the superblock.
c. The DRA is added as a named user with read and execute permissions to the default and access ACLs of the root directory of the encrypted volume.

Additionally, the permissions for the others entry are set to null. This recursively ensures that all further subdirectories and files created in the encrypted file system would automatically inherit these two entries in their access control lists.

It is possible to associate a separate cryptographic header with every encrypted file to enable per-file choice of different ciphers, modes, key sizes, IV methods and keyed hash algorithms. However, we believe this is an overkill of flexibility that decreases usability and necessitates the use of complex policies and configuration files. Utilizing common algorithm parameters for the entire volume provides the same level of security while making the system easy to use.

C. Mounting An Encrypted File System:

An encrypted file system is mounted by specifying the

CONFERENCE PAPER
II International Conference on
"Advance Computing and Creating Entrepreneurs (ACCE2013)"
On 19-20 Feb 2013
Organized by
2nd SIG-WNs, Div IV & Udaipur Chapter , CSI , IEEE Computer Society Chapter India Council ,
IEEE Student Chapter Geetanjali Institute of Technical Studies, Udaipur, Rajasthan, India

encrypt option. It is integrated with POSIX ACLs and hence the ACL mount option must also be specified. Also, the mechanism that is used to store private keys (whether smart cards are being used or not, for example) is specified as a mount option.

During mount, algorithm parameters are copied from the on-disk superblock of the underlying file system to the kernel's in-core superblock structure. The encrypted FSK is also read from the on-disk superblock, decrypted using the same mechanism used at the time of creation and copied into the in-core superblock.

File creation and access operations are shown in the following figures:

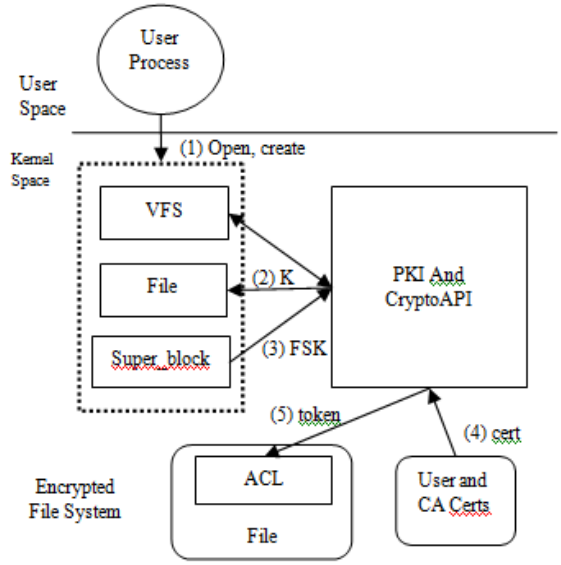

Figure 3: File operations: Creation

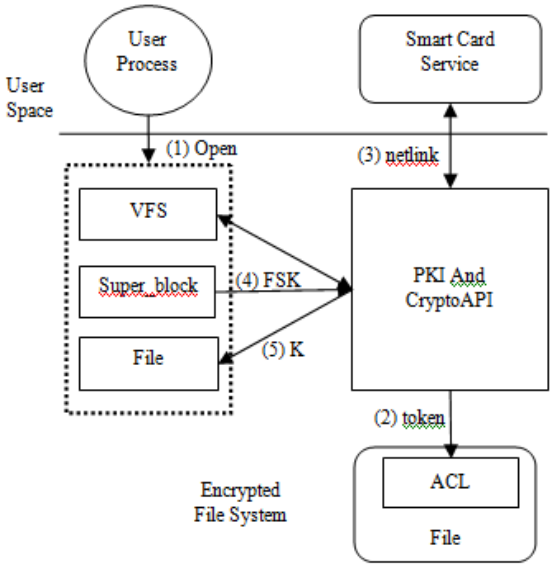

Figure 4: File operations: Access

The following actions are taken whenever a new file (or directory) is created in an encrypted file system:

A file encryption key k is randomly generated. It is put into the entry corresponding to this instance of open (or create) in the VFS open file structure. Also, the FSK is read from the kernel's in-core super block structure associated with the underlying volume. It is used to encrypt the FEK using the specified algorithm parameters.

a. The kernel determines the UID of the file's owner from the current process context. This is used to access the owner's certificate from the repository. The certificate is verified and then its public key is used to encrypt the result of the previous step. The resulting token is copied into the corresponding field of the owner's ACL entry.
b. The above step is repeated for all the users present in the default ACL inherited by the file (or directory) from its parent directory.

The actions taken when opening an existing encrypted file are as follows:

c. The kernel determines the UID of the current process context and checks the user's permissions to open the file using the appropriate ACL entry. If successfully verified, the corresponding token is pulled out of the ACL entry and decrypted using either the smart card or the user's private key acquired from the disk.
d. The file system key is read from the in-core super block and used to decrypt the result of the previous step.
e. If the user is genuine, we now have the original FEK k used to encrypt the file.
f. It is copied into the file structure corresponding to this call of open.

D. Reading And Writing File Data:

Other than read and write, file data may be accessed using the mmap system call. The 2.6 series kernels incorporate a unified page cache and bio infrastructure that provide a common interface to the disk regardless of the system calls used. Cryptographic file system takes advantage of these unified interfaces to hook in the encryption and decryption processes.

A file's contents are accessed after it has been opened. The FEK already present in the corresponding file structure is used to do encryption or decryption transparently.

Implementation issues such as locking and synchronization determine the exact placement of the encryption and decryption hooks in the kernel. The implementation effort has proceeded in an exploratory fashion and evolved towards the best alternative. A preliminary version plugged encryption and decryption at the page cache layer around the submit bh function.

This approach leads to individual bio requests being submitted for every file system block, thus causing a significant performance degradation of about 40%, as determined experimentally. This preliminary implementation approach is being discarded in favor of a design that uses the work queue interface, thus enabling the coalescing of multiple bio requests to avoid the aforementioned overhead.

Separate per-CPU kernel threads created in advance are executed in user process context. After encrypted data is read from the disk, the callback function executing in hard IRQ context merely enqueues the actual decryption job in the corresponding kernel

thread's work queue. The implementation of dm-crypt uses a similar design and integrating with it may be explored in the future.

V. CONCLUSION

Data security has emerged as a critical need in both personal and multi-user scenarios. The key challenge is to provide a solution that is easy to use for individuals as well as scalable for organizational environments. Most existing encrypting file systems do not meet the diverse requirements of security and usability, due to the lack of flexible key management, fine-grained access control and security against a wide range of attacks.

Our Conceptual Encrypting file System, provides a solution that is both secure and practically usable. We assume an attacker has the capability to launch attacks that are beyond the threat models of existing systems and proposes solutions to such threats. We make a crucial distinction between the kernel and user- space from a security perspective. Employing a completely kernel-space implementation enables us to avoid trusting the super user account and protect against various user-space attacks.

VI. ACKNOLEDGEMENT

This research paper is made possible through the help and support from everyone, including: parents, teachers, family, friends, and in essence, all sentient beings. Especially, please allow me to dedicate my acknowledgment of gratitude toward Dr. Tarun Shrimali and Ms. Ridhima Khamesra for their valuable advice and Guidance.

VII. REFERENCES

[1]. Matt Blaze, A Cryptographic File System for UNIX. In Proceedings of the ACM Conference on Computer and Communications Security, pages 9–16, 1993

[2]. Jean-Luc Cooke and David Bryson. Strong Cryptography in the Linux Kernel. In Proceedings of the Linux Symposium, pages 139–144, Ottawa, Canada, July 2003.

[3]. dm-crypt: a device-mapper crypto target for linux. website http://www. saout.de/misc/dm-crypt/.

[4]. Wolfgang Mauerer. Professional Linux Kernel Architecture, page 4-18

[5]. SmartK: a samrt card framework for Linux Kernel. //smartk.dia.unisa.it/.

[6]. Michael Austin Halcrow. eCryptfs: An Enterprise class Encrypted Filesystem for Linux. In Proceedings of the Linux Symposium, pages 201–218, Ottawa, Canada, July 2005.