Global Information Assurance Certification Paper

Cryptographic File Systems

Samuel Cote
SANS Security Essentials GSEC Practical Assignment Version 1.3
28 January 2002

Abstract

The majority (see [13]) of the top 20 Internet security flaws as reported by the SANS institute involve compromising superuser/administrator privileges, both under the Unix and Windows platforms. On top of this, [10] reports that 80% of computer access violations occur internally while 65% are caused by insiders to an organization. [11] and [12] both confirm that insiders seem to be the prime perpetrators of computer break-ins within an organization. This is significant in that gaining superuser access on a network or a node is easier done from the inside than remotely. A person working for the organization will probably already have a "legal" account on the organization's network, thus having more information on the network and its structure than a remote attacker might have. The major issue with such vulnerabilities is that once superuser access is compromised, all the data sitting on the machine (and possibly network, since alot of networks use a single trusted administrator account for all machines on a network or subnetwork, and Unix systems sometimes provide trusted administrative tools such as *rsh* and *rlogin* to navigate from machine to machine without having to re-authenticate yourself) on which the break-in occured is also compromised. The perpetrator of the break-in will have total access to the data. For an organization keeping a substantial amount of information available on its networks, such an intrusion can prove to be disastrous.

The use of a cryptographic file system can substantially decrease the possibility for an attacker to obtain access to sensitive information, even if administrator privileges have been compromised. While a cryptographic file system is not a totally fireproof solution to preserving data integrity in the case of unlawful access to superuser accounts, if properly used, it can help prevent data from being accessed by parties who normally would not have access to such data. This can be critical in a situation where sensitive information is kept on an organization's network system.

1.Introduction

The rest of this paper will survey various implementations of cryptographic file systems, their mechanisms and architectures, and how they can be used to enhance security within the right context.

2. Basic Principles

The *file system* is the data structure that describes how data is divided and stored as files on a storage medium. It keeps information about each file's attributes such as (depending on the file system type) file name, time stamps, access rights, file size, file type, owner, along with pointers to the actual data blocks that contain the file's data. Several file systems also offer extended functionalities such as journaling and network file sharing (such as Sun's NFS).

Cryptography is taken as the science of hiding data in plain sight using scrambling techniques.

A *cryptographic file system* extends the normal file system operations by providing crypting facilities to permit storing the file data (and, optionally, some attributes like the file name) in encrypted form.

Deniable cryptography is a cryptographic system that hides the amount and nature of encrypted data. An

attacker will only be able to see there is encrypted data on a device, not how much there is, and of what nature it is.

3. Storage Encryption

Storage-level encryption can be divided in three categories: traditional file encryptors, volume encryptors and cryptographic file systems.

File Encryptors

Traditional file encryption is done on a per-file basis using a user-provided passphrase (Phil Zimmerman's PGP is probably the most popular and widely used of these programs.) Accessing and storing an encrypted file will usually mean running the necessary tool each time, which makes routine use fairly unpractical, even if usage can sometimes be automated using shell scripts. This system is prone to human error, for example if a user forgets to re-encrypt the file after accessing it. It is also not very secure in a network/distributed environnement. The file itself will be vulnerable to interception while it is decrypted for access. And on a system where superuser access is compromised, the passphrase can be intercepted using key-logging, terminal snooping, or other techniques.

Although file encryptors are an acceptable solution for rarely-accessed files or e-mail communication encryption, using them for projects that involve sharing and routine access to sensitive information can prove to be extremely unpractical and tiresome to use. It is therefore not suited for a storage system.

Volume encryptors

Volume encryptors are usually implemented in the device driver layer and encrypt/decrypt data as it is transferred to and from the disk. They usually work by creating a "container" file in which a file system structure is built. The container file is encrypted, and contains all the user files just like on a normal file system. Access to the file system is done by creating virtual partitions that can be mounted and used just like a normal file system partition.

Such a mechanism is perfect for a single-user workstation. It offers granularity on a volume level, not on a file/directory level, and it is therefore not suggested for a group-sharing storage application. Also, individual file backup is impossible, as is normal administrative processes like integrity checking, file defragmentation and file system checks, since all the files are within the container file.

Implementations of this design include BestCrypt [14], PPDD [15], TorDisk and PGPDisk, among others.

Cryptographic File Systems

As previously mentioned, cryptographic file systems implement the crypting functions within the normal file system mechanism. This has several advantages over conventional cryptographic methods:

- **Transparent access,** with occasional requests for a passphrase and/or authentication;
- **Granularity** down to the directory and file level;
- **Prevents backup data theft**;
- **Makes system administration, file backups and maintenance possible by non-trusted personnel** since the data is protected at all times;
- **Prevents data theft in the case the attacker has physical access to the device**, or in the case the physical device gets stolen; with the advent of wireless networks and portable devices, more and more senstive information might end up on devices that can be easily stolen: laptop computers, handheld

devices, private digital assistants, etc.
- **Complements a secure-connection networking mechanism** by ensuring the data is also protected on the storage medium, not just while it is in transit;
- **Group file sharing** of sensitive information is made easier, since access to a particular ressource can be made possible using different keys for all individuals accessing the data, thus making key management more secure;
- **Audit trails** are easier to maintain, as most cryptgraphic file systems keep track of who accesses what data, and when it is accessed;
- **Different passwords** for different files and directories;
- Good implementations **support cleartext files and encrypted files on the same file system**, thus giving the possibility of enabling encryption on only a subset of the whole file system.
- **Deniable cryptography** can be implemented for maximum security. Consult [3] for an excellent treatise on deniable cryptographic file systems;

[5] provides an interesting list of real-world example applications in which cryptographic file systems can be used to significantly enhance security. These applications include:

- Private projects within an organization

Most organizations have projects that involve sensitive and private information. This information needs to be shared among the people that are part of a project, but the actual computer systems used by the group members will more than likely maintained by support personnel outside of the project group. By using a cryptographic file system to share and protect information, only the project members will be able to access the sensitive data, while all maintenance, backup, restoration and other administrative tasks necessary on the computer systems can be done by any other staff member without any threat for the data.

- Outsourced information storage

Organizations often use the services of other organizations to handle, manage and maintain their information storage infrastructure. This makes any data hosted by the storage provider vulnerable to any attack on the provider's systems, to any potentially malicious individual within the provider's organization, or to any of the provider's other customers who also have information hosted on the provider's systems. By making sure the information is encrypted as it is stored, an organization can safely let another organization handle the management of its data without fear for theft or unlawful access to the sensitive information.

- Outsourced intranet servers

In the case where a company or organization wants to outsource the management of its internal intranet servers to another organization, it has to make sure that the administrators of the servers do not pose a threat to the integrity of the data contained on the intranet servers. By using encrypting file systems to store data, the administrative tasks of running and maintaining the intranet servers can be carried out normally by an outside organization while still ensuring the safety of the data.

- Sharing of sensitive information between organizations

Business partnerships will often necessitate the sharing of private and sensitive information between the different entities involved. This is traditionally done by letting one organization access another organization's network system, for example thru lease lines behind firewalls or encrypted tunneled connections. This approach can lead to a higher level of vulnerability for the organization that provides access to its networks, and managing the lease-lines and secure connections will require additional network administration ressources. As certain implementations of cryptographic file systems provide the mechanism to safely protect information, even while it is in transit, sharing sensitive information becomes easier and

much safer for both sides or the partnership.

- Customers personal information

Just about every company keeps information about its customers. As this information is often sensitive (information about a customer with which a company has a major contract) or personal (contact and demographic informations about customers), it is critical to store it in a highly secure environement. For example, if a company does eCommerce thru a web site, offering a secure connection during personal information gathering does not ensure data privacy: the data also needs to be protected when it is stored. Cryptographic file systems again provide an elegant solution to the safe storage of such data.

- Keeping of medical records

Medical records are another type of information that could benefit from a safe and transparent storage system. They usually contain very personal information about an individual, and as such should be kept as private as possible. On the other hand, in the case of a medical emergency, it might be necessary to access and forward the necessary medical information rapidly to successfully treat the emergency (consider the case of someone travelling in another country.) A keeping system based on cryptographic file systems would permit the safe communication this information with the pertinent entities with minimal risk of interception.

4. Key Escrow

In the case where, within an organization, a particular individual is primarily responsible for the encrypted data, a *key escrow* [2] system can be implemented into the cryptographic file system to allow emergency access to the data even when the primary employee is absent or unavailable. Access can easily be granted, controlled, audited and revoked without compromising the organization's control over its information/data.

Smartcards constitute a good platform to implement key escrow systems for several reasons:

- They contain enough memory to store several keys along with audit information;
- They contain their own CPU, so a smartcard can be used as a standalone encryption/decryption engine (provided the smartcard has enough throughput to support access to potentially massive encrypted material);
- Data can be accessed without compromising the cryptographic keys security, as the keys never leaves the smartcard.

Note that reverse engineering on smartcards is possible. Thus, a cryptographic file system using smartcard technology to implement a key escrow system is limited in security by the smartcard's design to resist reverse engineering. Also, smartcard-based systems might not be appropriate for applications in which the data is highly sensitive or highly personal.

5. Implementations

Cryptographic file systems can be implemented in different ways. One way is to implement the mechanism directly in the kernel, or as a loadable kernel module (LKM) [9]. This provides tight integration with the system, maximum transparency, and stability. Other approaches include using the operating system's debugging interface to trap the system calls to read/write to storage, as a NFS client/server pair interface ([1], [7]), or using a file system extension (such as UFO) that permits transparent access to remote files thru the FTP and HTTP protocols [5].

Following is a review of some implementations of cryptographic file systems.

CFS (Cryptographic File System) ([1], [2])

CFS was developed by Matt Blaze at AT&T Laboratories for systems running Unix. Based on a NFS client/server architecture, it was designed as a research project and isn't currently updated. It features granularity on the directory level and implements a key-escrow system using smartcards. It doesn't support group sharing (each directory has a single key associated to it) and is not transparent since the user has to explicitly attach a directory to the tree before being able to access the data in the directory. The encryption scheme used is DES. There is also an extended version of CFS called by ECFS [17], developed by David Bindel, Chris Wells and Monica Chew, that adds encrypted data integrity and file-level granularity.

TCFS (Transparent Cryptographic File System) ([6], [7])

TCFS was developed at the University of Salerno, Italy. It also improves on the CFS architecture, removing the need to attach data to a directory and making the system more transparent. It is designed to work in kernel space by interposing itself between the Virtual File System layer and the storage file system (EXT2FS, UFS, NFS, etc.) It is currently implemented for Linux and (Net/Open)BSD and supports various crypting engines in the form of loadable kernel modules. Modules are provided for all major encryption schemes.

Cryptfs ([18])

Developed by Erez Zadok, Ion Badulescu & Alex Shender, this file system differs from the previous systems in that it is implemented at the level of kernel, instead of relying on a user-land based client/server architecture. It supports multi-user keys and makes sure that no permanent encryption data is saved on the disk. The first implementation was developed for Solaris, with subsequent versions being released for FreeBSD 3.0 and Linux 2.0. It is designed to run over already-existing file systems such as UFS, FFS and NFS, without requiring modifications to the file system or the operating system (as it uses loadable kernel modules in Linux and FreeBSD.) The cipher used is Blowfish.

StegFS (Steganographic File System) ([3], [4])

The original specification for this file system was developed in a paper by Ross Anderson, Roger Needham and Adi Shamir, and implemented in Linux (with a few minor differences from the original paper) by Andrew D. McDonald and Markus G. Khun. It was implemented by extending the ext2fs file system format, adding cryptographic functions but keeping the support for the unencrypted version of the file system, meaning that a normal ext2fs driver can access all the non-encrypted files on a StegFS-formated partition. The current implementation supports the Serpent and RC6 ciphers.

SFS (Secure File System) ([5])

Developed by J. Hughes, C. Feist, S. Hawkinson, J. Perreault, M. O'keefe and D. Corcoran under the Linux operating system. It was designed as a smartcard-based OS- and application-independent system, meaning that no recompilation is necessary either on the operating system or application side. It supports end-to-end encryption, and is implemented using a file system extension called UFO that provides transparent access to files on FTP and HTTP servers. The cryptographic functions are handled by the smartcard.The UFO system was extended to add crypting functions to be run on requested files (which gets downloaded by UFO on the local system from the remote server.) The Secure File System was developed as a proof-of-concept and still needs some improvement to be a fully functional and useable system.

RubberHose ([9])

RubberHose is a transparent and deniable cryptographic file system, and was developed by Julian Assange, Ralf P. Weinmann and Suelette Dreyfus. It is implemented as a loadable kernel module, with current support for Linux (2.2 kernel) and modules for NetBSD and FreeBSD nearing completion. It provides the DES, 3DES, IDEA, RC5, RC6, Blowfish, Twofish and CAST ciphers, alone or in combination (it is possible to "sequence" the different available encryption schemes.) It also supports the OpenSSL Library and time-based passphrases (that expire after a certain time). The deniable cryptography part is ensured by a random data blocks distribution on disk and resistance to statistical attacks as well as physical inspection attacks. It supports the UFS, FAT, FAT32 and ext2fs file systems.

ReiserFS v4 ([8])

ReiserFS is a file system designed by Hans Reiser. The new V4 version, which is due out on September 30th 2002, plans to support cryptographic functions along with multiple features that enhance security. The development is sponsored by DARPA, ApplianceWare and BigStorage, and will be included in the SE-Linux (Security Enhanced Linux, developed by the NSA) system (but usable on most Linux systems).

Windows 2000 Encrypting File System ([19])

The new release of the Microsoft Windows 2000 operating system includes integrated support for encrypted files on NTFS partitions. It is based on a public-key system and runs as an integrated service, so it is transparent to the user. The source code is also not provided, making inspection and analysis of the algorithms and architecture used fairly hard.

6. Benchmarks/Performance

CFS [2] demands 20-50% above the underlying filesystem workload. If a smardcard-based key-escrow system is used, the performance will depend on the smartcard's communication throughput speed, which varies with technology and brand/model. ECFS [17] reports similar results, with added overhead for the signing/hashing algorithms (tests used a dual Pentium-II 333MHz 64Mb memory system.)

On test made with a Pentium-II 233MHz system with 64Mb of RAM, the TCFS authors [7] report a fairly high (but acceptable) overhead because of the additional cryptographic functions. Cryptfs [18] removes some of the overhead by doing all its operations inside the kernel, and gets speed improvements of 12% to 52% over CFS.

The Linux implementation [4] of the the Steganographic File System on test using an AMD K5 P150 100MHz processor and 1Gb IDE disk shows performances close to ext2fs for normal (non-hidden) files, but hidden files induce a fairly higher processing cost. Read access is roughly 5 to 10 slower than ext2fs, while writing is 40 to 85 times slower. This is higher than the previous implementations since it uses a different process than the other file system to implement deniable cryptography.

Overall, results show that kernel-based implementations perform better than systems implemented as user-level daemons. This is because using a kernel-based approach will reduce the necessary state- and context-switching that is needed with user-level programs. Most papers report that almost all overhead comes from the cryptographic functions, so faster processors should minimize the performance cost of these systems.

7. Limitations

The main bottleneck of cryptographic file systems is the additional CPU processing required for the encryption and decryption process. The crypting routines have to be executed every time an access to

encrypted data occurs. The complexity of the processing will depend on the type of cipher (symmetric or asymmetric) and algorithms used, but in most case they will be substantial.

Some implementations also require an additional disk access to read access permissions or other data necessary to the crypting and authenticating processes. Hence running a cryptographic file system on a heavily loaded server, or a server containing data with a high availability requirement, is not a suggested approach at this point in time.

Interception of unencrypted data is possible if superuser access on a system is compromised and data is being accessed on that same system. The possibility of an interception will depend on how the cryptographic system is implemented (inside or outside the kernel), and how the rest of the operating system (mainly the memory management part) is built. Such an attack would involve gaining access to a process's (the one accessing the cleartext version of the encrypted data) memory space to access the data, or by "tapping" the data stream somewhere between the memory/application layer and file system layer. It is worthwhile to note that kernel memory space is harder to access than user-land memory space, making kernel-based implementations more secure.

Finally, as with all security-related devices and mechanisms that rely on keys and passwords, cryptographic file systems are only as secure as the password/key management policy surrounding the use of the system.

8. Possible Enhancements

Ideally, we would like the crypting functions to take as little additional processing and disk-access as possible. Kernel-based systems seems to offer the best performances in that respect, and the crypting functions could be sped up by using a hardware-based cryptographic processor, either on the motherboard or as a PCI (or other type) expansion card, removing the workload from the main processor. Such a card could also be used to encrypt communication channels.

Another very important point is that any system that manipulates encrypted data should also have encrypting enabled for the swap/virtual memory system [20]. The processes that manipulate the encrypted data will need to provide a plaintext version to the application/user that is using the data. Most modern operating systems provide a mechanism to swap out physical memory to disk to augment the available memory for processes. If a process accessing encrypted data (and thus manipulating a plaintext version of it) gets swapped out, the unencrypted information will end up in the virtual memory backstore. Such information can remain available for reading after the process stops, even across multiple reboots. Encrypting the virtual memory ensures that no plaintext versions of encrypted data and user passphrases end up in the virtual memory system.

9. Conclusion

Cryptographic file systems are still a fairly new approach to data security. While there are not that many mission-critical grade systems available (at least, outside of military circles) to organizations at this point in time, several systems that are being developed look very promising, and the overall field should see more developement in the coming years, as alot of organizations shifted their policies to devote more ressources and attention to computer systems security.

The important thing to remember is that cryptographic file systems do not constitute by themselves a complete security solution. They can significantly enchance security if properly used, but security remains a process more than a product. A system will only be as secure as its weakest link, however strong the rest of the links are.

10. References

[1] Blaze, Matt "A Cryptographic File System for Unix." *Proceedings of the First ACM Conference on Computer and Communications Security*, Fairfax, VA, November 1993.
URL: http://www.crypto.com/papers/cfs.pdf

[2] Blaze, Matt "Key Management in an Encrypting File System." *USENIX Summer 1994 Technical Conference*, Boston, MA, June 1994.
URL: http://www.crypto.com/papers/cfskey.pdf

[3] Anderson, Ross, Needham, Roger & Shamir, Adi "The Steganograhpic File System"
URL: http://www.cl.cam.ac.uk/~mgk25/sfs3.ps.gz

[4] McDonald, D. Andrew & Kuhn, G. Markus "StegFS: A Steganographic File System for Linux" 2000
URL: http://www.cl.cam.ac.uk/~mgk25/ih99-stegfs.pdf

[5] Hughes, James, Feist, Chris, Hawkinson, Steve, Perrault, Jeff, O'Keefe, Matthew & Corcoran, David "A Universal Access, Smart-Card-Based, Secure File System". Febuary 8 2000
URL: http://www.securefs.org/sfspaperfeb.pdf

[6] Catuogno, Luigi "Transparent Cryptographic File System of BSD - A Primer" April 15 2000.
URL: http://tcfs.dia.unisa.it/BSD/tcfsbsd-primer.pdf.gz

[7] Cattaneo, Giuseppe, Catuogno, Luigi, Del Sorbo, Aniello, & Persiano, Pino "The design and Implementation of a Transparent Cryptographic File System" *USENIX Annual Technical Conference 2001 - Freenix Track* 2001
URL: http://www.tcfs.it/docs/freenix01.ps

[8] Reiser, Hans "ReiserFS V4" 2001
URL: http://www.namesys.com/v4/v4.html

[9] Dreyfus, Suelette "The Idiot Savant's Guide to Rubberhose" 2001
URL: http://www.rubberhose.org/current/src/doc/maruguide.pdf

[10] "Computer Security Facts and Statistics from Harris Corporation" March 29 2001
URL: http://www.softwareshelf.com/downloads/misc/statfacts.html

[11] "SAIC Final Data Security Report to the California Electronic One-Stop Committee" 10 May 1999
URL: http://www.sjtcc.cahwnet.gov/SJTCCWEB/ONE-STOP/security/Final.htm

[12] Teach, Edward "Look Who's Hacking Now" *CFO Magazine* Febuary 1 1998
URL: http://www.cfo.com/Article?article=1510

[13] SANS Institute "The Twenty Most Critical Internet Security Vulnerabilities - The Experts Consensus" November 15 2001
URL: http://www.sans.org/top20.htm

[14] BestCrypt Homepage
URL: http://www.bestcrypt.com

[15] PPDD Homepage
URL: http://linux01.gwdg.de/~alatham/ppdd.html

[16] Colohan, Chris, Rosenberg, Chuck & Steffan, Greg "Secure Sharing with Satan's File System, from Selected Reports: Fall 1997 Software Systems Course" *Technical report CMU-CS-98-103, School of Computer Science*, Carnegie Mellon University, April 1998.
URL: http://reports-archive.adm.cs.cmu.edu/anon/1998/CMU-CS-98-103.ps

[17] Bindel, David, Chew, Monica & Wells, Chris "Extended Cryptographic File System"
URL: http://www.cs.berkeley.edu/~dbindel/oceanstore/ecfs.pdf

[18] Zadok, Erez, Badulescu, Ion & Shender, Alex "Cryptfs: A Stackable Vnode Level Encryption File System" *Columbia U. CS TechReport CUCS-021-98*, July 1998
URL: http://www.cs.columbia.edu/~ezk/research/cryptfs/cryptfs.pdf

[19] Microsoft Corporation "Encrypting File System for Windows 2000" July 01, 1999
URL: http://www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp

[20] Provos, Niels "Encrypting Virtual Memory" *USENIX Security 2000*
URL: http://www.openbsd.org/papers/swapencrypt.ps