Crocus: A Steganographic Filesystem Manager

HIOKI Hirohisa
Center for Promotion of Excellence in Higher Education, Kyoto University
Yoshida-Nihonmatsu-cho, Sakyo-Ku, Kyoto, 606-8501, Japan
hioki@i.h.kyoto-u.ac.jp

ABSTRACT

Cryptographic filesystems are widely used to protect private files. It is, however, impossible to hide the existence of private information by such filesystems. Steganographic filesystems attempt to address this problem by embedding files imperceptibly into containers. In most steganographic filesystems ever proposed, files are embedded into containers those apparently randomized. Their existence would, however, imply that they include hidden files. This paper presents a new steganographic filesystem manager called Crocus. When a filesystem is to be hidden, it is embedded separately into a set of innocent-looking containers piece by piece. When the filesystem is to be used later, it is reconstructed from the pieces. We can resize or destruct the filesystem if required. Since more than one containers can be used for one filesystem, we can build filesystems those large enough. A prototype system of Crocus has been developed for Linux and a preliminary experiment was performed. The result indicates the effectiveness of the framework of Crocus.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*information hiding*

General Terms

Design

Keywords

Filesystem manager, Steganography

1. INTRODUCTION

Cryptographic filesystems are widely used to protect private files. We can make files unreadable by storing them in such filesystems; but even so, it is still evident that we have unreadable encrypted files. Steganographic filesystems[1, 3, 4, 6, 7, 8, 9] attempt to address this problem by embedding files imperceptibly into containers like disk partitions or files. The files embedded there apparently disappear from the local disks.

In most steganographic filesystems ever proposed, files are hidden in containers those apparently randomized and logically divided into more than one layers. This type of steganographic filesystem is said to have the property of plausible deniability. Assume that we are threatened to disclose all the hidden files under a filesystem. In such a case, we can open a decoy layer that includes only not-really sensitive files and say we have disclosed all. The more sensitive layers are then kept secret, since the threatener cannot know whether such layers exist from outside. This model of plausible deniability is indeed useful. It is, however, unusual that we have randomized volumes those not used for any purpose. Their existence might arise suspicion that they include hidden files.

Steganography literally means the technique of hiding secrets behind something else that camouflage their existence. In this sense, we should use common containers with their own contents. Many steganographic methods have been proposed for embedding secret stuff into containers like graphical images, sounds and texts[5, 8]. Unfortunately, they work only for a single container, which limits severely the sizes of filesystems we can build.

This paper presents a new steganographic filesystem manager called Crocus. It is a descendant of AshFS[4]. Crocus allows us to embed one filesystem separately into a set of innocent-looking containers piece by piece. When the filesystem is to be used, it is reconstructed from the pieces. We can resize or destruct filesystems if required.

Since more than one containers can be used for embedding one filesystem, we can build filesystems those large enough. This framework is a key feature for constructing steganographic filesystems those not based on randomized volumes.

Although AshFS has been designed under the same framework, the number of containers that can be assigned to one filesystem is actually limited. The flexibility of AshFS is also limited in the sense that filesystems managed under it are not resizable. The containers should be selected from a single filesystem in AshFS, while Crocus is not subject to such a limitation.

A prototype system of Crocus has been developed for Linux. A preliminary experiment was performed on the prototype system. The result indicates the effectiveness of the framework of Crocus.

The rest of this paper is organized as follows. Section 2 presents the basic concepts of Crocus. Section 3 outlines the prototype system. The experimental result is shown

ASIACCS'07, March 20-22, 2007, Singapore.

in Section 4. Finally, Section 5 concludes this paper with directions of future work.

2. BASIC CONCEPTS

This section presents the basic concepts of Crocus. The overview of filesystem management methods is given first. Next, the procedure for embedding payloads (pieces of a filesystem image and others) is described. We will then see how to find the layout of containers of a hidden filesystem.

2.1 Filesystem Management Methods

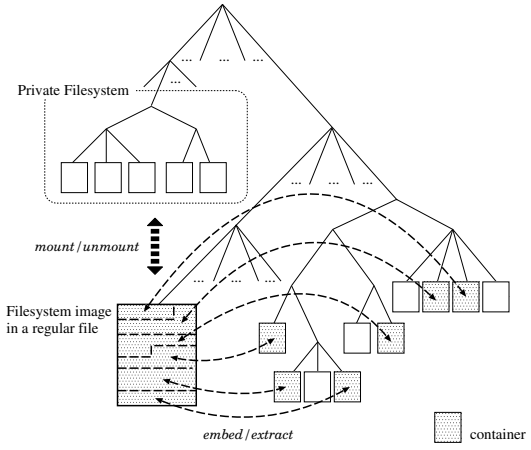

Figure 1: The filesystem model under Crocus

Crocus can create, open, close, resize or destroy filesystems those under its control. Figure 1 shows the filesystem model under Crocus. The private filesystem in the figure represents a filesystem managed by Crocus.

We can use various files as containers. Different types of container are processed in a uniform way through the *virtual steganographic interface module* (VSIM). When a container is given to VSIM, it selects an appropriate *steganographic engine* for the container. Steganographic engines are registered with Crocus as plugins for VSIM. They directly interact with containers for embedding or extracting payloads. A file is accepted as a container when a suitable steganographic engine is registered and its capacity is larger than a predetermined threshold V_T.

We can create a new filesystem by giving containers and a password to Crocus. Crocus then requests the system OS to build a new filesystem image within a regular file and to mount it[1]. The size of the image is computed from the total capacity of the containers. Since we can allocate more than one containers for one filesystem, we can make a filesystem as large as we like if we can provide enough containers and have a space to write the file holding its image.

While a filesystem is mounted, it is fully operated by the system OS. We can thus read or write files stored there as usual. A filesystem which is mounted is said to be in the *active* state.

Closing a filesystem brings it to the *hidden* state. Crocus requests the system OS to unmount it first. Right after it is unmounted, the filesystem image is embedded separately into containers piece by piece. The file holding the filesystem image is then randomized and deleted[2]. The filesystem now becomes imperceptible (and temporarily unavailable). Note that the imperceptibility of the hidden filesystem depends on the steganographic engines.

We can open the hidden filesystem later to bring it back to the active state and use it again. On opening the hidden filesystem, all the pieces are extracted from the containers and the filesystem image is reconstructed from them. Once the filesystem image is reconstructed, we are ready to mount and use it. In order to perform this reconstruction, we must know how to find the containers and how to arrange them. This problem is addressed in Section 2.3.

It is possible to resize existing filesystems by adding or removing containers. Shrinking is performed only when filesystems are not damaged.

We can destroy hidden filesystems to wipe out them. Destruction is performed (without deleting containers) by filling all the containers with dummy payloads steganographically. Since the filesystem image is destroyed completely, not only the files in the hidden filesystem but also the filesystem itself is revoked.

2.2 Payload Embedding Procedure

On embedding, a payload is passed to VSIM from the core system of Crocus. VSIM encapsulates the payload as a *packet*. It is composed of a magic string, a string which represents the size of the payload and the payload itself.

A packet will not be embedded directly into a container, or attackers might find the container and read payloads inside. In order to protect payloads, first, each packet is encoded using a pseudo-random number generator and partitioned into small blocks. Those blocks are then shuffled according to a pseudo-random permutation and are passed to an appropriate steganographic engine. The steganographic engine is then embedded the (shuffled) blocks into the container. The password assigned for the filesystem is used for encoding and shuffling. This protection method is called *scrambling* in this paper.

2.3 Finding Layout of Containers

In Crocus, a hidden filesystem is embedded across a set of containers. In order to open and use the hidden filesystem later, we must find all of its containers and arrange them in the correct order to reconstruct the filesystem image.

Assume that the filesystem image is embedded piece by piece into the containers $C_0, C_1, \ldots, C_{n-1}$ in this order. In Crocus, we can open a hidden filesystem by giving the password assigned for it and the *root container* C_0, i.e. the first one of the containers.

Let $\mathrm{id}(C_i)$ be the identifier of the container C_i. We can find the layout of containers via the root container if the sequence of identifiers $\mathrm{id}(C_1), \mathrm{id}(C_2), \ldots, \mathrm{id}(C_{n-1})$ is embedded along with the filesystem image. This sequence of identifiers is called *layout information of containers* (LIC).

The LIC is embedded into the root container, at least partially. Note that the identifier $\mathrm{id}(C_i)$ should be embedded into a preceding container $C_j (j < i)$. Let $V(C_i)$ be the capacity of the container C_i and $\mathrm{pre}(j)$ be the prefix of the LIC up to the end of the identifier of the container C_j. We can then find that the whole LIC can be restored on extraction

[1]Crocus thus works on an OS under which we can use a regular file as a virtual disk.

[2]Here, randomization is performed to prevent unsolicited recovery of the deleted file.

if the following is satisfied:

$$1 \leq \forall j < n,\ \mathrm{pre}(j) \leq \sum_{i=0}^{j-1} V(C_i). \quad (1)$$

Since the LIC can extend for more than one containers, we can assign containers for one filesystem as many as we want. On the other hand, AshFS has the restriction that the LIC must always be embedded entirely into the root. This limits the number of containers assignable to one filesystem.

3. PROTOTYPE SYSTEM

A prototype system of Crocus has been developed for Linux. In the prototype system, we can use graphical image files and audio files as containers if they are in lossless-compression or in uncompressed formats such as PNG, BMP and WAV. Steganographic engines based on the simple LSB method[5] are employed both for graphical image files and audio files. Filesystems are formatted as ext2fs and accessed via loopback devices. The standard e2fsprogs utilities are used for creating or resizing filesystems.

The LIC of a filesystem is derived from the pathnames of containers. The identifier $\mathrm{id}(C_i)$ for the container C_i becomes either its absolute pathname or relative pathname from the directory where the container C_{i-1} resides. The shorter one is selected to make the LIC shorter.

4. AN EXPERIMENTAL RESULT

A preliminary experiment was performed on the prototype system. A filesystem was created from 38 containers. Low 2 bits of graphical image files and low 4 bits of audio files were used for embedding respectively. The total data size (not file size) and the total capacity of the containers were 20412344 bytes and 5103085 bytes respectively.

The filesystem image was created in the unit of 4k bytes, which is the page size of Linux. The block size for the scrambling was set to 512 bytes. The capacity threshold V_T for containers was set to 2k bytes.

The size of the filesystem created from the containers was 5095424 bytes. The sizes of the LIC, packet headers and unused fragments were 718 bytes, 439 bytes and 6504 bytes respectively. We can find that the ratio of the image size of the filesystem to the total capacity is about 99.8%. Most of the capacity was thus devoted to the filesystem image. The average PSNR of graphical image files was 44.1dB and that of audio files was 74.8dB. Noise added to the containers through embedding was not perceptible. This result demonstrates that Crocus enables us to obtain steganographic filesystems those fairly large and indicates the effectiveness of the framework of Crocus.

5. CONCLUSION AND FUTURE WORK

The steganographic filesystem manager called Crocus is presented in this paper. A filesystem managed by Crocus is built in a regular file and is mounted as a normal filesystem. It can be hidden steganographically into a set of innocent-looking containers while it is not in use. Crocus enables us to have hidden filesystems those not based on randomized volumes.

We can open a hidden filesystem by giving the root container and the password for it. We can make a filesystem as large as we like if we can provide enough containers and a space to place the file holding its image. We can extend or shrink filesystems by adding or removing containers. Destruction of filesystems is also allowed.

A prototype system of Crocus has been implemented for Linux. A preliminary experiment was performed on the prototype system and the result indicates the effectiveness of the framework of Crocus.

Crocus hides images of filesystems only and does not hide filesystems in the active state. It is worth considering to take another design model where filesystems are always separately hidden in containers. If we develop a new driver which interacts with containers for reading or writing hidden files directly, filesystems are not entirely visible anymore even when we use them and extra files holding filesystem images are not required as well.

Hidden filesystems created by Crocus are fragile. A filesystem corrupts immediately if one of its container is lost. Providing robustness for hidden filesystems is another important issue to make them more suitable for practical use.

The existence of Crocus may attract the attention of attackers. A possible work around is to embed Crocus itself into a set of containers and to prepare a small boot-strapping program for extracting it. On installing the boot-strapping program, we should name it appropriately and adjust its code carefully through an obfuscation method[2] not to leave its signature.

6. ACKNOWLEDGEMENT

This research was partially supported by the Ministry of Education, Culture, Sports and Technology, Government of Japan, Grant-in-Aid for Scientific Research, 16700096, 2004.

7. REFERENCES

[1] R. Anderson, R. Needham, and A. Shamir. The steganographic file system. In *Information Hiding: 2nd International Workshop*, pages 73–82, 1999. LNCS 1525.

[2] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, University of Auckland, New Zealand, 1997.

[3] S. Dean. Freeotfe. http://www.freeotfe.org/.

[4] H. Hioki. A scattered hidden file system. In *Proceedings of Pacific Rim Workshop on Digital Steganography 2004*, pages 89–95. Kyushu Institute of Technology, 2004.

[5] N. F. Johnson, Z. Duric, and S. Jajodia. *Information Hiding: Steganography and Watermarking – Attacks and Countermeasures*. Kluwer Academic Publishers, 2001.

[6] A. D. McDonald and M. G. Kuhn. Stegfs: A steganographic file system for linux. In *Information Hiding: 3rd International Workshop*, pages 463–477, 2000. LNCS 1768.

[7] H. Pang, K.-L. Tan, and X. Zhou. Steganographic schemes for file system and b-tree. *IEEE Transactions on Knowledge and Data Engineering*, 16(6):701–713, 2004.

[8] SecureStar Ltd. Drivecrypt. http://www.securstar.com/products_drivecrypt.php.

[9] TrueCrypt Foundation. Truecrypt. http://www.truecrypt.org/.