Digital Watermarking of Virtual Machine Images

Kumiko Tadano, Masahiro Kawato, Ryo Furukawa, Fumio Machida, Yoshiharu Maeno

Chapter 18

DIGITAL WATERMARKING OF VIRTUAL MACHINE IMAGES

Kumiko Tadano, Masahiro Kawato, Ryo Furukawa, Fumio Machida and Yoshiharu Maeno

Abstract The widespread use of server and desktop virtualization technologies increases the likelihood of unauthorized and uncontrolled distribution of virtual machine (VM) images that contain proprietary software. This paper attempts to address this issue using a platform-independent digital watermarking scheme applicable to a variety of VM images. The scheme embeds a watermark in the form of files in a VM image; the watermarked VM image is identified based on the embedded files. To reduce the possibility of discovery by an attacker, the names of the embedded files are very similar to the names of pre-existing files in the VM image. Experiments indicate that the approach is fast and accurate, with average turnaround times of 24.001 seconds and 7.549 seconds for watermark generation and detection, respectively.

Keywords: Digital watermarking, virtual machine images

1. Introduction

In modern enterprise computing there is a growing trend toward consolidating virtual machines (VMs) in the server and client sides to reduce costs and enhance portability and security. Such environments require the means to export VM images to test software and to backup VMs. For example, Amazon's Elastic Compute Cloud [1] and Simple Storage Service [2] provide the command `ec2-download-bundle` to download VM images from a data center to local computers [3]. Unfortunately, this technology also facilitates the unauthorized dissemination of VM images containing proprietary software.

One approach for addressing this issue is to use host-based intrusion detection systems such as TripWire, AIDE and XenFIT [8]. These

systems monitor unauthorized file system changes to detect malicious activity involving VMs. However, it is difficult to identify VM images after they have been distributed outside of an enterprise network (e.g., using peer-to-peer file sharing software).

Another approach is to implement strict access control and copy control of VM images. However, these controls often hinder the legitimate use of VMs.

Therefore, it is necessary to address two issues: (i) identify VM images even after they have been illegally distributed; and (ii) facilitate legitimate use of VM images. Digital watermarking of VM images can address both these issues. However, as we discuss below, watermarking techniques used for audio and video files are not applicable to VM images.

Digital watermarking technologies typically modify redundant or unused digital content (e.g., inaudible frequency ranges for audio files) to embed watermarks. In the case of audio and video files, modifying the original information for digital watermarking produces imperceptible effects when playing the files [7]. In contrast, modifying a VM image can cause boot failures when the watermarked image is executed. Additionally, data on the watermarked VM is frequently changed because of software updates, logging, etc. Unlike an audio or video file whose content is unchanged, a VM image is essentially variable; consequently, in order to identify the VM image, the image has to be watermarked after every change.

Data hiding techniques [5] can be employed for watermarking VM images. Data can be hidden in various locations:

- Areas marked as not in use by the partition table.
- Extended file attributes such as alternate data streams.
- Unused portions of the last data units of files (slack space).
- Reserved i-nodes that are not used by the operating system.
- Portions that are excluded during consistency checking of a journaling file system.
- Files hidden via steganography using special file system drivers [6].

A major deficiency of existing data hiding techniques is the lack of platform-independence: the techniques work on specific file systems (e.g., NTFS), operating systems (e.g., Red Hat Linux) and OS kernel versions (e.g., Linux 2.2.x). It is difficult, if not impossible, to apply

these techniques to VMs in heterogeneous environments where multiple file systems, operating systems and OS kernel versions are used.

To address this issue, we propose a digital watermarking scheme for VM images that is independent of file systems, file formats, operating systems, OS kernel versions and hardware. The scheme embeds a watermark derived from the names of the files present in a VM.

2. Basic Concepts

Our digital watermarking scheme for VM images uses file names as a watermark, not the contents of the files. A unique identifier is created for each VM image and a watermark corresponding to the identifier is embedded in the form of files in the file system of the VM image. These embedded files are called "watermark fragment files."

If the names of the watermark fragment files are randomly created, they would be readily distinguishable from the names of the pre-existing files on the VM, enabling an attacker to identify and subsequently remove the watermark fragment files. Thus, the watermarking scheme makes the fragment files difficult to detect by creating fragment files with names that are similar to pre-existing files in the VM and embedding the files in random directories in the VM. A secure database is used to store the identifier of each VM image and the corresponding watermark fragment files.

3. Design and Implementation

This section describes the design and implementation of the digital watermarking scheme.

3.1 System Components

The system has three components: (i) a watermark generator; (ii) a watermark detector; and (iii) a watermark information database. Figure 1 illustrates the watermark generation and detection processes.

- **Watermark Generator:** The watermark generator employs the names of the watermark fragment files as the watermark for a VM image. It creates a watermarked VM image by embedding the watermark fragment files in the VM image and stores the generated file names in the watermark information database. The VM image to be watermarked is created by copying the template VM image file. Since the template VM images are used as the original data for watermarking, these images should be securely managed. The reason is that an attacker who obtains the VM image template

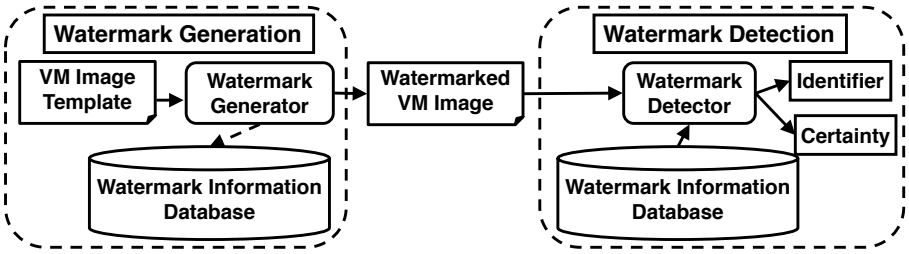
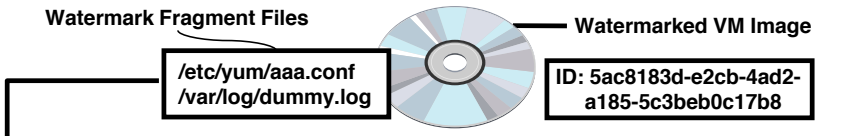


Figure 1. Watermark generation and detection.

would be able to compute the watermark fragment files as the difference between the template VM image and the watermarked VM image files. Details of the watermark generation algorithm are presented in Section 3.2.



Watermark Information Database Table

ID	VM Name	Owner	Watermark Fragment Files
5ac8183d-e2cb-4ad2-a185-5c3beb0c17b8	foo-desktop	foo	/etc/yum/aaa.conf, /var/log/dummy.log
42be8a40-5e6d-4677-84fb-e387f5638264	bar-desktop	bar	/usr/local/bin/cmd, /etc/dummy.conf
...	...	...	...

Figure 2. Sample records in the watermark information database.

- **Watermark Information Database:** The watermark information database (WI-DB) stores the identifier of each VM image and its attributes. The identifier for a VM image is a universally unique identifier (UUID). The attributes include the names of the watermark fragment files, VM owner, etc. (Figure 2).

  Only authorized users should be permitted to read and write WI-DB records. An attacker with read/write access could alter WI-DB records or identify and delete the watermark fragment files embedded in a VM image.

- **Watermark Detector:** The watermark detector identifies a VM using the watermark fragment files embedded in the target VM image based on WI-DB records. When an unauthorized leak of a VM image is suspected, the administrator may perform the detection process to identify the VM image and its owner.

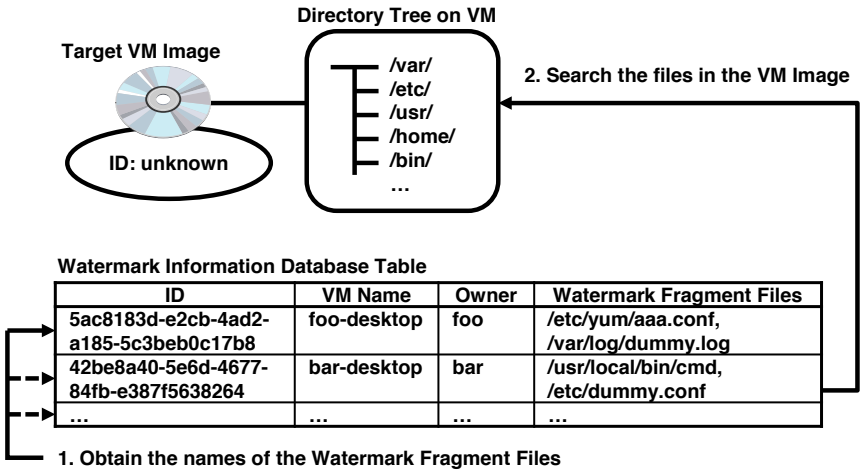


Figure 3. Watermark detection.

Figure 3 illustrates the detection process. If the identifier of the target VM can be guessed, the watermark generator looks up the corresponding WI-DB record. Otherwise, the watermark detector compares the watermark fragment file names in WI-DB records with those in the target VM image. The watermark generator then outputs a certainty value of the identifier of the target VM image. The certainty value is computed as the percentage of the names of the watermark fragment files in the WI-DB record that match file names in the VM image. Note that some watermark fragment files may have been deleted or renamed on purpose or by accident.

3.2 Watermark Generation Algorithm

This section briefly describes the process of generating the watermark fragment files for the target VM image. The algorithm has five steps.

- **Step 1:** Input the parameters required for watermark generation.
- **Step 2:** Number the directories in the target VM image in dictionary order.
- **Step 3:** Select the directories to embed watermark fragment files.
- **Step 4:** Generate the names of the watermark fragment files to be created.
- **Step 5:** Generate the contents of the watermark fragment files for the directories selected in Step 3.

Table 1. Directory code list.

Number	Directory Name
1	/
2	/etc/
3	/etc/init.d/
4	/etc/ldap/
5	/etc/skel/
6	/etc/ssh/
7	/etc/sysconfig/
...	...

Step 1 (Input Parameters for Watermark Generation): The following input parameters are required for watermark generation:

- **Number of Watermark Fragment Files:** The larger the number of watermark fragment files, the more tamper-resistant is the watermark. However, this increases the time required to generate watermark fragment files.
- **Excluded Directories:** Some directories (e.g., temporary directories) should be excluded because files in these directories change frequently.
- **Excluded Suffixes:** These are suffixes that should be excluded from the names of the watermark fragment files.
- **Bit Length:** The bit length is a parameter used by the Blum-Blum-Shub (BBS) algorithm [4] to generate cryptographically-secure pseudo-random numbers. This parameter affects the computational time and security, and is set to 1024 bits in our experiments.

Step 2 (Number the Directories in the Target VM Image): The watermark generator searches all the directories in the target VM image recursively and generates a "directory code list" (Table 1). The excluded directories identified in Step 1 are not numbered. The directory code of a template VM image can be reused when new watermarked VM images are created from the same template VM image because they have the same directory tree structure. This reduces the processing time.

Step 3 (Select Directories to Embed Watermark Fragment Files): The watermark generator produces pseudo-random numbers for the watermark fragment files input in Step 1. The values of the

random numbers are limited to the range of directory codes. The watermark generator extracts the names of directories corresponding to the generated random numbers using the directory code list generated in Step 2. The BBS pseudo-random number generation algorithm is used to make the random numbers difficult for an attacker to predict. Although this algorithm is computationally intensive, the experimental results presented in Section 4.2 indicate that the overall performance of the watermarking scheme is acceptable.

Step 4 (Generate the Names of Watermark Fragment Files): The watermark generator creates the names of the watermark fragment files that are embedded in the directories selected in Step 3. Portions of the names of pre-existing files in the directories are modified to create file names that are similar to those of the pre-existing files. A file name is generated according to the following steps.

- **Step 4.1:** Get the names of existing files in the target directory. Only regular files (not subdirectories, hidden files, etc.) whose suffixes are not to be excluded are retrieved.

- **Step 4.2:** Enumerate all the substrings corresponding to the file names. Each substring is obtained from the head (i.e., not including the suffix) of a file name retrieved in Step 4.1. The result of this step is the union of substrings for all the file names. For example, if the target directory has a file named `abc.txt`, the possible substrings are: `abc`, `ab` and `a`.

- **Step 4.3:** Compute "similarity groups" for each extracted substring. A similarity group is a group of files in the target directory that meets the following conditions: (i) the file name starts with the substring (prefix) generated in Step 4.2; (ii) all the files have the common suffix (e.g., `.txt`); and (iii) a similarity group contains at least two files. For example, if there are five files in a target directory {`s1.txt`, `s2.txt`, `s3.dat`, `s4.dat`, `s5.conf`} and `s` is the substring, then two similarity groups can be computed: {`s1.txt`, `s2.txt`} and {`s3.dat`, `s4.dat`}.

- **Step 4.4:** Compute the D_{sim} score for each similarity group:

$$D_{sim} = c_1 * l_p + c_2 * n_f - c_3 * l_d$$

  where l_p is the length of the prefix of the similarity group; n_f is the number of files in the similarity group; l_d is the mean value of the difference between the length of the file name (excluding the

suffix) and l_p in the similarity group; and c_1, c_2, c_3 are parameters that are set to one. The similarity group with the highest D_{sim} score is called the "prototype file group."

- **Step 4.5:** Create the name of the new watermark fragment file. One or more random letters are added to the tail of the prefix of the selected prototype file group. The length of the added letters (l_t) is the length of the file name randomly selected from the prototype file group (excluding the suffix and prefix). Finally, the suffix of the files in the prototype file group is added to the file name. For example, if the prefix is `sample` and the prototype file group is {`sample.dat`, `sample-bak.dat`} and if `sample-bak.dat` is selected, then $l_t = 4$ and a possible name is `sampledbha.dat`.

Step 5 (Generate the Content of the Watermark Fragment Files): The watermark generator creates the content and attributes corresponding to each watermark fragment file. The content of a watermark fragment file is a randomly-generated byte sequence. Attributes of a watermark fragment file are determined according to the following rules. For timestamps (creation/modification/access) and file size, the watermark generator assigns the mean values of the files in the prototype file group to the new watermark fragment file. For other attributes such as ownership and permission, the values corresponding to a randomly selected file in the prototype file group are used.

3.3 Implementation

The digital watermarking scheme was implemented in Java 1.5. The VMware Server virtualization software was employed. The command `vmware-mount.pl` was issued to the VMware Server to mount the file system on the VM image for embedding watermark fragment files. The VM image of any file system or operating system, including Windows and Linux, can be watermarked. The WI-DB was implemented using MySQL 5.0.

4. Experimental Setup and Results

This section describes the experimental setup used to evaluate the watermarking algorithm. Also, it presents the experimental results related to watermark generation and detection.

4.1 Experimental Setup

A test environment was created to evaluate the performance of the watermarking scheme. The environment included one physical host (Ta-

Table 2. Test environment.

Physical Host		
CPU	Pentium 4 1.73 GHz Processor	
Memory	1 GB RAM	
Host OS	Ubuntu Linux 8.04 Server	
VMM	VMware Server 2.0	
	Guest VMs	
Guest OS	**Ubuntu Linux 8.04 (JeOS edition)**	**CentOS 5.1 (default)**
VM Image Size (`.vmdk file`)	183 MB	2,750 MB
Directories	1,120	7,429

ble 2). Two template VM images of different sizes were used to clarify the impact of size on the watermarking scheme. The VM images used were an Ubuntu Linux 8.04 Server JeOS edition (minimum configuration for virtual appliances) and a default installation of CentOS 5.1.

4.2 Experimental Results

This section presents our experimental results related to watermark generation and detection.

Watermark Generation The VM image templates of Ubuntu and CentOS were copied and a total of 10 and 100 watermark fragment files were embedded in the two VM images. Each experiment was repeated 10 times and the average turnaround times were calculated.

Table 3. Turnaround times for watermark generation.

	Av. Generation Time (sec)		**Av. Copying Time (sec)**
	10 Files	**100 Files**	
Ubuntu Linux 8.04	13.121	24.001	6.903
CentOS 5.1	168.382	177.270	142.284

Table 3 presents the turnaround times for copying VM images and generating watermarks. The experiments used previously-created directory codes of template VM images because the codes are generated only the first time that the template VM images are used. The results indicate that the time for watermark generation excluding the time for

Table 4. Generated watermark fragment files.

File Names
/usr/share/zoneinfo/Pacific/kQGidyI
/etc/console-tools/v2uQizvontwL8
/lib/modules/2.6.24-16-virtual/kernel/sound/usb/snd-usb-INc.ko
/usr/lib/klibc/bin/hso.shared
/usr/lib/perl/5.8.8/IO/Socket/vS9jkE.pm
/usr/lib/python2.5/wsgiref/hG272nT.pyc
/usr/share/debconf/jAGX7mQ.sh
/usr/share/zoneinfo/Mexico/irQ7KA
/usr/share/zoneinfo/posix/Chile/5XwjA

copying is essentially independent of the size of the template VM image. The turnaround times drop when faster storage devices are employed.

Table 4 lists the generated watermark fragment files. The file names `snd-usb-INc.ko` and `hso.shared` are relatively difficult to distinguish from the pre-existing files in the target directories. In contrast, file names such as `kQGidyI` are easily distinguishable. In general, when there are many files in the target directory whose names have common extensions and long common substrings, the generated file name(s) tend to be indistinguishable. For example, the file `snd-usb-INc.ko` belongs to */lib/modules/2.6.24-16-virtual/kernel/sound/usb/* whose pre-existing files are `snd-usb-audio.ko` and `snd-usb-lib.ko`. Meanwhile, if only a few file names have common substrings/extensions in the target directory, the generated file name(s) tend to be contrived.

Watermark Detection Watermark detection experiments were conducted to identify 10 and 100 watermark fragment files from the watermarked Ubuntu 8.04 and CentOS 5.1 VM images.

Table 5. Turnaround times for watermark detection.

	Av. Detection Time (sec)	
	10 Files	**100 Files**
Ubuntu Linux 8.04	7.524	7.549
CentOS 5.1	7.526	7.573

Table 5 presents the average turnaround times for watermark detection based on ten repetitions. The results indicate that the time taken to detect the watermark is almost independent of the size of the watermarked VM image and the number of watermark fragment files.

5. Conclusions

The digital watermarking scheme for VM images is independent of file systems, file formats, operating systems, kernel versions and hardware. Also, experiments demonstrate that watermark generation and detection are both fast and effective.

Our future work will focus on enhancing the tamper-resistant characteristics of the watermarking scheme. Several attacks could be devised to remove or modify VM image watermarks. For example, watermarks could be detected using a machine learning algorithm such text clustering [9] to discriminate embedded files from pre-existing files based on file name string features. Another attack could use information from elsewhere in the VM such as the i-node numbers of files. Files installed at a given time are assigned successive i-node numbers; a file with an i-node number that is out of sequence could correspond to an embedded file. Other problems to be investigated include having the contents of embedded files resemble those of pre-existing files, and augmenting the scheme with other data hiding approaches to enhance tamper resistance. Finally, our research will investigate the application of the watermarking scheme to other digital content such as tar and zip archives.

References

[1] Amazon Web Services, Amazon Elastic Compute Cloud, Seattle, Washington (aws.amazon.com/ec2).

[2] Amazon Web Services, Amazon Simple Storage Service, Seattle, Washington (s3.amazonaws.com).

[3] Amazon Web Services, ec2-download-bundle, Seattle, Washington (docs.amazonwebservices.com/AmazonEC2/dg/2006-10-01/CLTRG-ami-download-bundle.html).

[4] L. Blum, M. Blum and M. Shub, Comparison of two pseudo-random number generators, in *Advances in Cryptology: Proceedings of Crypto 1982*, D. Chaum, R. Rivest and A. Sherman (Eds.), Plenum, New York, pp. 61–78, 1982.

[5] K. Eckstein and M. Jahnke, Data hiding in journaling file systems, *Proceedings of the Fifth Annual Digital Forensic Research Workshop*, 2005.

[6] A. McDonald and M. Kuhn, StegFS: A steganographic file system for Linux, *Proceedings of the Third International Workshop on Information Hiding*, pp. 463–477, 2000.

[7] F. Perez-Gonzalez and J. Hernandez, A tutorial on digital watermarking, *Proceedings of the Thirty-Third IEEE International Carnahan Conference on Security Technology*, pp. 286–292, 1999.

[8] N. Quynh and Y. Takefuji, A novel approach for a file-system integrity monitor tool for a Xen virtual machine, *Proceedings of the Second ACM Symposium on Information, Computer and Communications Security*, pp. 194–202, 2007.

[9] T. Segaran, *Programming Collective Intelligence: Building Smart Web 2.0 Applications*, O'Reilly, Sebastopol, California, 2007.