New Protection of Kernel-level Digital Rights Management in Cloud-based Consumer Electronics Environments

Woei-Jiunn Tsaur\textsuperscript{1} and Lo-Yao Yeh\textsuperscript{2}

\textsuperscript{1}Department of Information Management, Da-Yeh University, Changhua, Taiwan
\textsuperscript{2}Network and Information Security Division, National Center for High-Performance Computing, Taichung, Taiwan

Abstract – Controlling and managing rights of digital contents has become very critical in cloud-based consumer electronics devices. The kernel-level digital rights management (DRM) software can offer stronger protection of digital contents. For effectively preventing unauthorized copying, the rootkit stealth technologies may be employed in consumer electronics (CE) environments to conceal kernel-level DRM driver. Therefore, to stop unauthorized users from deleting the DRM software by employing anti-rootkit tools to remove the rootkit, this paper presents new rootkit-based stealth technologies to reinforce DRM driver in cloud-based CE environments. The proposed new driver-hidden rootkit stealth technologies can successfully evade detection and removal of well-known rootkit detectors in cloud-based CE environments. In summary, the contributions of this paper are that in cloud-based CE environments the proposed novel stealth technologies can be used to effectively reinforce the kernel-level DRM software for ensuring the rights of legitimate consumers and providing forceful protections for copyright owners.

Keywords: Cloud service, consumer electronics, digital rights management, consumer security system, Linux.

1 Introduction

With the advent of new technologies such as cloud computing, it seems to cause problems of piracy even more than before. A huge amount of transactions of digital contents are expected under cloud computing environments. And the cloud computing environment is a place where many computers are available everywhere throughout the physical world connected seamlessly to the information systems. Therefore, it will be a more critical issue to control and manage rights of digital contents. Piracy and illegal distribution of digital contents are severe issues. Digital Rights Management (DRM) \cite{1}-\cite{6} aims at protecting digital contents from being abused through regulating the usage of digital contents. The DRM scheme is a digital protection technique that protects and manages the access rights of digital contents. It can prevent the confidential information of a digital content from unauthorized usages by illegal users. Although each DRM system may have its different DRM implementation and infrastructure, the basic DRM process is the same, which usually involves three parties: content provider, DRM technology provider (i.e. cloud server in cloud-based consumer electronics (CE) environments), and consumer device. The implementation of a DRM controller in the kernel of an operating system, instead of implementing it at the application layer potentially provides some great benefits \cite{7}. The main advantage is that it should be possible for any application to access protected works, as the main underlying protection is provided by the operating system and not the individual applications. Furthermore, it also means that DRM protection can be offered to any data types, not just multimedia, and remain transparent to any user-space application trying to access the DRM protected contents. This means that DRM can be used as a privacy enhancing mechanism for consumers, who can determine the exact access control rules for their own private data.

As stated in the literature \cite{8}, \cite{9}, rootkit is a stealth technology, and the intent with which this technology is used determines their malicious or otherwise legitimate purpose. The same technology used by rootkits is also used in security software such as firewalls and host-based intrusion prevention systems to extend the protection of the operating system. Therefore, the rootkit technologies may be employed in CE devices to conceal the DRM software for preventing unauthorized copying \cite{10}. However, Tsaur's scheme \cite{10} cannot be adopted in cloud-based CE environments because his proposed approach did not completely consider the properties of virtual machines and operating systems in cloud-based CE environments. In addition, cloud environments offer particularly attractive malware targets as they incorporate vast numbers of computing resources and high network bandwidths, and are increasingly becoming the operational home to many high-valued software systems and services. Attacks targeting clouds can provide significant chances to obtain control over resources and extract proprietary information. Thus, how to control and manage rights of digital contents has been becoming very critical in cloud-based consumer electronics environments.

In order to prevent unauthorized users from removing the rootkit of concealing kernel-level DRM driver by employing anti-rootkit tools in cloud-based CE environments, this paper is to propose new driver-hidden rootkit stealth technologies for strengthening the DRM driver in protecting against the illegal distribution and consumption of copyrighted digital contents. In cloud-based environments, though many companies or organizations have been developing rootkit detectors to the public and undoubtedly they can detect known rootkits effectively, they cannot foresee what the result is when meeting unknown rootkits. In this paper, the proposed
unknown rootkit stealth technologies are constructed in Linux-based cloud operating systems, and have been verified that it can successfully evade detection and removal of a variety of well-known rootkit detectors. The contributions of this paper are mainly twofold. Firstly, when consumers use their CE devices to connect to cloud servers, the proposed new rootkit stealth technologies can be employed to extend the protection of the kernel-level DRM driver in cloud-based consumer entertainment environments, which can be a great inspiration to DRM software makers to effectively ensure the rights of legitimate consumers and provide strong protections for copyright owners. Secondly, the stealth tricks of the proposed subtle rootkits can be a great inspiration to defenders who need to effectively strengthen the legitimate uses in cloud-based service environments.

The remainder of this paper is organized as follows. Section 2 surveys current rootkit stealth and detection technologies. Section 3 presents the system design for developing new rootkit stealth technologies in Linux-based cloud operating systems. Section 4 depicts the experimental results of testing the proposed rootkit’s stealth ability in Linux-based cloud operating systems. Finally, some concluding remarks are included in the last section.

2 Related work

There are essentially two different technologies that a rootkit can use to hide computer resources. One is hooking that intercepts the requests of accessing resources. The other is Direct Kernel Object Manipulation (DKOM) that manipulates the data used by operating systems to keep track of resources. The oldest kernel mode rootkit [8] uses table hooking to alter System Service Descriptor Table (SSDT) to hide processes, drivers, files, etc. Although it is a simple, stable and efficient method, it is easily detected by current rootkit detectors [11]. Hunt and Brubacher [12] introduced Detours, a library for intercepting arbitrary binary function. Later, this method is also applied by a rootkit, which replaces the first few instructions of a specific function with “jump” to point to the rootkit’s code instead of targeting system tables. The aforementioned is named inline hooking. But VICE [8], a heuristic detector, is created to detect hookers no matter table hooking or inline hooking. In order to enhance inline hooking, brilliant rootkit makers combine a polymorphic technique [13] whose purpose is to generate different appearances of a piece of code. These appearances may look different but have the same functionality. On the other hand, Butler et al. [14] used DKOM to target EPROCESS, a kind of kernel data structures to record information related to a process, to alter an affiliated doubly linked list, and let the desired processes be hidden. When using DKOM, rootkit makers need to clearly understand the data structures in kernel, but it is more furtive than using hooking [8], [15]. The DKOM technique was first used in the FU rootkit and then used in FUTo to hide their drivers [8], [9]. In 2007, the DKOM-based Unreal rootkit was created and shown off that all of the famous detectors cannot detect it. However, at present several well-known detectors are capable of effectively detecting the above-mentioned three driver-hidden rootkits.

As for identifying rootkits, there are two main approaches to develop rootkit detectors. The first detection approach targets hiding mechanisms by detecting the presence of API hooking [15], [16]. It is similar to the signature-based detection [17], and thus it cannot catch unknown rootkits whose signatures of hiding mechanisms do not exist in its signatures repository in advance. The second approach targets hiding behaviors by detecting any discrepancies between the original and the fake. It collects resources information from two different storage places, and then compares each other to find rootkits. This approach is to belong to the cross-view rootkit detection [18]. It is noticed that in this approach both targeted information cannot be modified simultaneously by rootkits, otherwise a detector using this approach cannot distinguish the differences between the two places storing targets and then it must be useless. Although this method has the drawback, it does not need to maintain a signature database as used in the signature-based detection method.

3 System design

In this section, new rootkit stealth techniques which have abilities to escape well-known anti-rootkit tools are proposed to hide the driver-format DRM software in effectively protecting against the illegal distribution and consumption of copyrighted digital contents. This paper discloses six places, some of which may not be known by anti-rootkit developers, to hide driver information in Linux-based cloud operating systems. The proposed new rootkit stealth technologies are composed of six tricks which will be presented in the following Items A-F, respectively.

A. Removing the signature of ELF (Executable and Linkable Format) image

The ELF file format is executable in operating systems environments and can be executed in multiple platforms. Almost all executable files (also including kernel mode drivers) are to use the ELF file format which inherits the characteristics of the COFF (Common Object File Format) file format in UNIX operating systems. The ELF file format contains four main parts [19]: ELF Header, Program Header
Table, a variety of Sections, and Section Header Table, as shown in Fig. 1. Driver programs are loaded to memory via initializing RAM disk system function, and therefore their complete ELF images exists in memory after loading. In order to avoid detectors to scan the ELF files, the characteristics of the ELF images must be removed. The method of removing the ELF images is to first load the ELF file to memory, and then find the list of the shared library link, including ELF32_Shdr[0], …, and ELF32_Shdr[n-1] in Section Header Table as shown in Fig. 1. Afterwards, the corresponding symbol link and library in the file system can be found from ELF32_Phdr[0], …, and ELF32_Phdr[n-1] in Program Header Table, and then the value of ELF Header is set to zero.

B. Removing Object Drivers and Object Devices from Object Directory

In the internal of operating systems kernel, the most fascinating part is objects. They contain all kinds of resources that are queried by kernel functions. The program of object management is responsible to manage objects. All of them are kept in a tree of Object Directory whose definition is shown in Fig. 2. The Object Directory is established with several HashBuckets. Each one points to an Object Directory Entry structure whose definition is shown in Fig. 3. In Fig. 3, it can be found that the Object member refers to an object, and ChainLink member points to another Object Directory Entry. Most of Object Drivers have at least one Object Device pointing to themselves, so both of them are needed to consider when a driver-hidden rootkit is made. Thus, the purpose of hiding can be achieved by exploring the whole Object Directory to find the desired Object Drivers and Object Devices to hide.

C. Removing Object Drivers from driver’s Object_Type

An Object_Type defines the common properties of the same kind of objects as shown in Fig. 4. Each kind of objects has a dedicated Object_Type. An Object_Type has a List_Entry data structure which keeps all of the same kind of Object_Creator_Info. The definition of Object_Creator_Info is described in Fig. 5. Here the type of objects is referred to Object Driver whose definition is shown in Fig. 6. And every object body is immediately preceded by an Object_Header structure whose Type member points to the Object_Type. Therefore, the loaded rootkit driver can be exploited to get its Object Driver, then move to its Object_Header to get the pointer to the Object_Type, and finally check its TypeList member to find the desired Object Drivers to hide.

D. Removing Object Devices from device’s Object_Type

It is the same as the method described in Item C, except for different kind of objects. Here the type of objects is referred to Object Device whose definition is shown in Fig. 7. First, through the loaded rootkit Driver the Object Device can be found. After getting the Object Device, an Object_Type can also be found. Finally, its List_Entry is traversed to locate the desired Object Devices to hide.
E. Removing drivers from Linux loadable kernel modules

A Linux loadable kernel modules (LKM)s structure whose definition is shown in Fig. 8 can be effectively employed to hide drivers. In this trick, the loaded rootkit driver can be checked to get its Object Driver’s DriverSection, and further find and traverse LKMs to get the desired driver addresses to hide.

F. Altering Object Driver’s appearance

The targeted Object Driver appearance is modified to let it look different as compared to a normal one. This method tries to escape signature-based detectors. For example, if the value stored in the offset 0x000h of an Object Driver should be 0x04, then it can be altered with a random value to accomplish the purpose of stealth.

4 Experimental result and analysis

In Section 3, the six stealth technologies of the proposed new rootkit in Linux-based cloud operating systems have been depicted. In the following, the experimental results of testing the proposed rootkit’s stealth ability in Linux-based cloud operating systems (Linux Mint 15 Cinnamon (Olivia)) are demonstrated by the following two phases: (1) rootkit loading operations, (2) test and analysis of rootkit stealth ability.

A. Rootkit loading operations

The proposed Linux-based rootkit named rootkit_dyu is a driver format and executed in a cloud computing service where multiple virtual machines are co-located on the same physical server. In such systems, physical resources are transparently shared by the virtual machines belonging to multiple users. For the rootkit loading operations, the proposed rootkit is installed but its stealth functionality is not invoked, as shown in Figs. 9 and 10. In Fig. 10, the rootkit driver name “rootkit_dyu” can be clearly found after it is loaded, and therefore it attests that the proposed rootkit is successfully loaded into the Linux-based cloud operating system.

B. Test and analysis of rootkit stealth ability

In order to provide greater flexibility in the testing process, the GUI (Graphical User Interface) interface is designed to test the stealth ability of the proposed rootkit, as shown in Fig. 11. In Fig. 11, since the six hiding techniques have been implemented in the proposed “rootkit_dyu” drivers, testers can check off their needs of the concealment mechanism to deploy a variety of different type of rootkit.

When the six stealth tricks of the proposed rootkit “rootkit_dyu” are checked off in Fig. 11, “rootkit_dyu” disappears in the Linux-based cloud operating system, as
shown in Fig. 12, and thus proves that it has launched the hiding feature. As stated in the literature [8], [20]-[27], it can be found that a variety of detectors are highly effective for identifying rootkits. Therefore, the prestigious rootkit detectors introduced by the literature [8], [20]-[27] have been chosen to effectively test the stealth ability of the proposed rootkit driver. When the proposed rootkit is installed but its stealth functionality is not invoked, all of the detectors have listed it. When its stealth functionality has been invoked, all of the well-known detectors cannot detect the presence of the proposed rootkit in cloud computing environments.

It can be concluded that why all of the tested detectors cannot detect the proposed rootkit driver should be the following reasons. One is that some detectors cannot detect the rootkit with the abilities of removing the signature of ELF image. Another is that some detectors employ memory scan with predefined signatures, but they cannot recognize hidden Object Drivers with an abnormal object appearance. The other is that some detectors do not completely check whether the data structures of Object Directory, Object Driver, Object Device and LKMs may be modified, and thus the rootkit with the tricks of modifying the aforementioned data structures can avoid the heuristic-based detectors.

Fig. 9. The proposed rootkit “rootkit_dyu” is installed.

Fig. 10. The stealth functionality of the proposed rootkit “rootkit_dyu” is not invoked, and therefore it is shown in the Linux-based cloud operating system.

Fig. 11. The GUI for testing the stealth functionality.

Fig. 12. After the stealth functionality of the proposed rootkit “rootkit_dyu” is invoked, it disappears in the Linux-based cloud operating system.

5 Conclusions

In this paper, novel Linux-based rootkit stealth technologies are presented for enhancing kernel-level DRM driver in preventing the confidential information of digital contents from unauthorized usages by illegal users in cloud-based CE environments. The proposed new driver-hidden rootkit technologies executed on Linux-based cloud operating systems has successfully evaded the well-known anti-rootkit detectors, and thus can be effectively used to prevent unauthorized users from removing the rootkit of concealing the DRM driver by employing anti-rootkit tools in cloud-based CE environments.

To the best of the author’s knowledge, there is no literature exploring the rootkit-based technologies for enhancing the kernel-level DRM driver in CE environments at present, so this paper is the first attempt to develop kernel-level DRM
protection technologies against unauthorized usages of digital contents in cloud-based CE environments. The proposed technologies are valuable for extending the protection of DRM software, and can be a great inspiration to cloud-based DRM software makers to effectively improve the current techniques of defending against the illegal distribution and consumption of copyrighted digital contents. Furthermore, this study also inspires defenders to effectively strengthen the legitimate uses in cloud service environments by the proposed subtle hiding tricks.

Acknowledgement

This work was supported by the Ministry of Science and Technology of Republic of China under contract numbers NSC 101-2221-E-212-006-MY3 and MOST 103-2622-E-212-009-CC2.

References


