Vol.2009-ARC-183 No.30
Vol.2009-OS-111 No.30
2009/4/24

VMM-based Detection of Rootkits that Modify File Metadata

Kenji Kono,[†1,†2] Pushkar R. Rajkarnikar,[†1]
Hiroshi Yamada[†1,†2] and Makoto Shimamura [†1]

Kernel-level rootkits are posing an immense threat to any computer systems. They operate inside operating system kernels and manipulate crucial kernel data structures to bypass and evade anti-malware security tools. Combined with kernel-level rootkits, other malware such as bots, viruses and spyware becomes stealthy and difficult to detect. The focus of this paper is on typical kernel-level rootkits that falsify the contents of file systems to hide the presence of malware. In this paper, we propose a software based system that leverages virtual machine technology. Unlike traditional approaches, our system does not rely on signatures. Instead, it relies on the rootkit behavior that file system contents are falsified; it detects a mismatch between the file system view from user-level processes and that from the virtual machine monitor running under the operating system. The experimental results demonstrate that our system successfully detects real world kernel-level rootkits and the overhead of the system is reasonable.

1. Introduction

Rootkit technology has evolved considerably over the years. From the earlier user-land rootkits to the recent kernel-level rootkits, the attackers have used highly sophisticated and novel techniques in order to subvert a computer system. Along with the development of new detection systems, the attackers have also developed their own versions of improved malware technologies to defeat any available detection systems and this race of cat and mouse does not seem to be over pretty soon.

In fighting against these kinds of rootkits, any anti-malware system that runs in the same level as the malware may not be able to detect the presence of malware. These anti-malware systems may be subverted by the malware before they can detect the presence of malware. Virtualization technology can solve this serious shortcomings of these existing anti-malware systems. Virtualization technology provides virtual machines (VMs) with strong isolation among them and prevents any access to or interference with other VMs or the VMM.

In this paper, we present RootkitLibra which leverages virtualization technology and resides in VMM. RootkitLibra being a behavior-based anti-malware system, focuses on a fundamental filesystem related behavior of a typical kernel-level rootkit. We observe that most kernel-level rootkits employ file or directory hiding technique as well as changing various filesystem metadata such as size, owner, access time etc. of some crucial system files but mask those changes from the applications and system administrators.

RootkitLibra basically uses a cross-view technique to detect the above mentioned behavior of any rootkit. In a cross view technique the same piece of system information is checked from two different views, one from a trusted vantage point and the other from the untrusted location. If the two views result in different outcome, then we can point out the presence of malware within the system. RootkitLibra assumes a networked file system to get the trusted view of the filesystem in VMM. RootkitLibra gets the untrusted view of filesystem information from return values of the system call.

We implemented RootkitLibra in Xen Hypervisor and successfully detected 8 real world Linux kernel-level rootkits. RootkitLibra imposes a small runtime overhead of about 1.3% when tested with the postmark[14] filesytem benchmark and less than 5% when tested against some basic filesystem micro benchmark tests.

2. Overview of Kernel-level Rootkits

Kernel-level rootkits are the second generation of rootkits and are relatively new and advanced compared to their first generation counterparts. These rootkits employ various stealthy techniques in order to hide files and directories and subvert the kernel of the system. They inject code directly to the kernel memory area. Once they install themselves, along with hiding themselves, they hide other malware, subvert crucial kernel entities and also provide backdoors for entry into the kernel in future. The installed rootkit can hide various processes, files and

†1 Keio University
†2 CREST, JST

network connections from the system administrators as well as from any user land applications. Users and administrators cannot trust any results received from the kernel or any system security tools or file system integrity tools that run as an application on top of the subverted kernel. As a result, most of the host-based tools that aim to detect kernel level rootkits are rendered ineffective.

Kernel-level rootkits use various attack vectors. They primarily aim to subvert some crucial kernel objects like system call table, interrupt descriptor table (IDT) or virtual filesystem (VFS) and insert their own version of those objects. With these growing techniques to sabotage the kernel, anything in the kernel can be the next target of the rootkits. New kernel-level rootkits might target subsystems such as the scheduler, network stack, hardware drivers etc. Common attack vectors used by kernel-rootkits are as follows[25]:

Intercepting System Calls A rootkit can target a system call table and replace entries of the system call table with its own version of certain system calls. Rootkit can also modify the system call handler and insert its own code immediately before and after the call to system calls and change the values returned by the system calls before it is given to applications. Roootkits such as Adore, Knark, Override[2] employ this attack vector.

Hooking and patching Kernel Jump Tables OSes use jump tables as the entry point to various kernel functions. A common jump table that is the target of rootkits is IDT. As in case of system call table, rootkit can modify the addresses of various interrupt handler routines in IDT to execute its own version of the code. Rootkits can also modify the first few instructions of interrupt handlers that push the handler code's address to stack and jump to error_code routine from where the handler is invoked.

Modifying Kernel Memory A rootkit can exploit kernel objects such as Linux's /dev/kmem which provides access to the kernel memory. With this interface, rootkits can redirect system calls to its own version without modifying the original system call table[7]. Rootkits such as SuckIt, Mood-NT, Superkit[2] use this technique.

Intercepting calls to VFS Kernel VFS layer is an abstraction layer on top of a concrete filesystem. This layer lets a uniform access to the underlying filesystem. Adore-Ng[2] rootkit exploits this layer and redirects the handler routines of VFS to its own version.

An attacker can inject malware code inside the kernel as loadable kernel modules (LKM), device drivers or by directly manipulating the kernel memory via standard virtual devices such as Linux's /dev/kmem or /dev/mem or /proc file system.

3. Design Goals

Guard Filesystem Integrity: The primary objective of RootkitLibra is to detect kernel-level rootkits that hide files or modify filesystem metadata. We observe that most kernel-level rootkits try to sabotage the filesystem by hiding various files, directories or changing various filesystem metadata such as size, owner, access time, links etc. Thus, RootkitLibra primarily focuses on the integrity of filesystem to detect the presence of rootkits. So, its first goal is to guard the integrity of filesystem of the monitored OS.

Robust Detection System: Traditionally, anti-malware systems have resided together with the host system. Though these kinds of anti-malware systems may be useful for user-level rootkits and some other types of earlier malware, they are not at all acceptable in case of kernel-level rootkits. Since kernel-level rootkits compromise various critical kernel entities, any applications that run on top of the infected kernel cannot be trusted. As a consequence, a kernel-level rootkit detection systems that reside in host has every chance of being tampered by the underlying rootkit in the kernel. So, the next design goal of RootkitLibra is that its integrity should not be questioned or doubted and that it must reside in a location which is firmly isolated from the possible malware attack.

Independent of Specific OS version: RootkitLibra should be generic and independent of any specific versions of OS. This makes it applicable to a wide array of OSes and increases its usability. There have been many researches that try to detect kernel-level rootkits from "outside-the-box". But, they assume substantial knowledge about the OS kernel implementation. For example, VMwatcher[11] assumes we can keep track of process tables, Copilot[13] assumes the intimate knowledge of the kernel layout like the module structure of monitored OS and so on. These kinds of dependency on the specific knowledge about "not-so-stable" OS internal structures reduce the portability and applicability of the detection

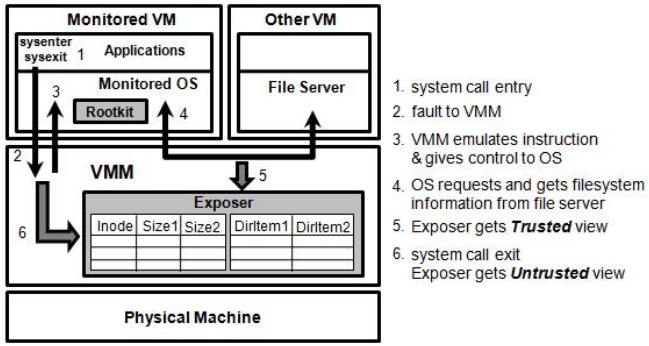

Fig. 1 Architecture of RootkitLibra

system. So, the third goal of RootkitLibra is that it should not rely on any kind of specific OS internal structures which makes it more generic and portable.

4. RootkitLibra

RootkitLibra monitors the basic behavior of rootkits i.e. hide files/directories or modify filesystem data/metadata but hide those modifications from user-land applications and administrators. Whenever any rootkit attempts to commit that behavior, either during its installation or after being installed, RootkitLibra catches that behavior by finding the inconsistencies between the VMM-level view (trusted view) and user-application-level view (untrusted view) of the same piece of filesystem data. Even though the design principles of RootkitLibra is applicable to other OSes, the following description of the design is based on Linux on intel x86 machine.

4.1 Getting Trusted View

Trusted view is the view as seen from VMM layer. VMM layer has access to various low-level states such as disk blocks, memory pages etc. of VMs running on top of it. These low-level but reliable states can be reconstructed in VMM to infer high level states of VMs such as files, processes etc. Since RootkitLibra is monitoring the integrity of filesystem, we need to get these high-level filesystem infomation of the monitored OS in VMM. We can employ a number of alternative techniques to get this information. One way would be directly reconstructing files/directories from low-level disk blocks of monitored VM accessible to VMM. Another alternative would be using a networked file system such as Network File System (NFS) protocol or Common Internet File System (CIFS) protocol as a filesystem of the monitored OS. These protocols work by exchanging network packets that use high-level filesystem information during the communication between client and server. Out of these two approaches, the first approach requires low-level observation of disk blocks and is pretty cumbersome. So, to avoid this cumbersome work, RootkitLibra employs the latter approach and uses a networked file system for the OS it is monitoring. The use of a networked file system necessitates the existence of another server machine which hosts the filesystem of the VM we are monitoring. To eliminate the need for another separate physical machine, we can setup this file server machine on the same physical machine as another VM.

Even though the use of networked file system obviates the need to reconstruct low-level VM states to high-level VM states, RootkitLibra still has to do some extra work. During the runtime of the monitored OS, there will be many filesystem operations such as read, write, append, create or delete filesystem objects. Since the filesystem resides in a network, in the course of performing these operations, the OS must fetch the filesystem data from the networked file server. During this process, these filesystem data make their way through the virtual network driver in VMM as network packets. RootkitLibra which resides in VMM, closely observes and dissects each of these network packets going in and out of the monitored VM to get the high-level filesystem information of monitored OS in VMM. This view of the monitored filesystem is a clean and trusted view which is yet to be processed by any kernel-level services of the monitored OS which might be infected with rootkits.

The observation and dissection of the network packets in VMM yields various filesystem data/metadata such as inode, size, owner, group, time related information and number of directory entries etc. of all the files and directories accessed during the run time of the monitored OS. To use these filesystem information, RootkitLibra maintains two tables. First, the file-table which stores inode and size of each accessed file and second, the directory-table which stores the inode, size and directory entries of each accessed directory. RootkitLibra uses these

tables to check the consistency of the filesystem data between the trusted and untrusted views.

4.2 Getting Untrusted View

To get untrusted view, we need to get values returned from kernel services such as system calls of the untrusted kernel. Since the kernel may be compromised, we cannot modify the monitored kernel to get these pieces of information. Moreover, modification does not make sense as we cannot rely on any values returned from the untrusted kernel. In view of these problems, we leverage the virtualization technology. Whenever a user-application requests a service from the kernel, it does so through some system calls.

RootkitLibra filters system calls by observing the value of `eax` register whenever there is a transition to VMM due to system call in guest OS. Whenever there occurs a file-related system call, RootkitLibra extracts the system call inputs and outputs of those system calls. It then uses these values to get the filesystem metadata such as the inode, size, number of directory entries etc. These filesystem metadata are untrusted because they are the OS-level view which are returned from the services (system calls) of the kernel of the monitored OS which might be infected with rootkits. RootkitLibra then inserts these data in two tables it maintains for consistency check. The first table is the file-table which is a table of inode and filesize of each accessed file and the second table is the directory-table which is a table of inode, size and number of directory entries for each accessed directory. During the process of extracting these high-level filesystem semantics of monitored guest OS from VMM, RootkitLibra uses the value of CR3 register to infer the OS process-level semantic in VMM[24].

4.3 Raising an Alert

RootkitLibra collects the same piece of filesystem information from two sources: 1) the trusted souce which is from outside the monitored OS and 2) the untrusted source which is from inside the monitored OS. While collecting these information, RootkitLibra first comes across the trusted view because when filesystem related system calls are invoked, those system calls need to fetch the data from the network file server and all the fetched data make their way through VMM as network packets. So, RootkitLibra first extracts filesystem data at this point. For each file, it extracts inode number and file size and for each directory it extracts inode number, size and number of directory entries and store them in file-table and directory-table respectively. Next, when the filesystem data arrive in guest OS, it performs the requested operation and the system call exits and the control faults to VMM. At this point, RootkitLibra uses the system call outputs and with the help of the knowledge of system call interface, it collects the untrusted view of the filesystem data. Next RootkitLibra compares the collected untrusted filesytem data with the trusted filesystem data collected and stored earlier in file-table and directory-table. Finally, RootkitLibra raises an alert if there is inconsistency between those two view.

Hidden files/directories or modifications made to filesystem data/metadata would be visible in VMM level view but remain hidden in VM level view because VMM level view is an external view which is not returned from the kernel services of the monitored guest OS whereas VM level view is the internal view which is returned from the services of infected kernel of the monitored OS.

5. Evaluation

In this section, we describe the evaluation of RootkitLibra by running it against some real world rootkits to test its effectiveness. We also performed the overhead experiments and analyzed its findings regarding the runtime overhead it incurred.

5.1 Evaluation against real world rootkits

We have evaluated our system with 8 real world linux kernel-level rootkits that are publicly available[2]. All these kernel-level rootkits can effectively subvert any vulnerable system successfully. Since RootkitLibra primarily relies on the filesystem integrity in order to detect the rootkits, these rootkits do hide files and/or directories in the system or alter certain metadata of files in the monitored system but hides the change from the administrators. Since all of the rootkits were not compatible for a paritcular kernel version, we implemented them in different versions of paravirtualized Xen and Linux. We experimented adore-0.42 and rial in Xen 2.0.7 with linux 2.4.30 as monitored system. Similarly enyelkm, override and adore-ng-0.56 was tested on Xen 3.0.2-2 as hypervisor and linux 2.6.16 as monitored system. We implemented mood-nt, superkit and suckit2priv in Xen 3.0.1 and linux 2.6.12. A very few minor modifications were required to get some of these rootkits installed in our test system.

Rootkit	Attack Vector	Target	Detected
adore-0.42	LKM	Sys call table	Yes
rial	LKM	Sys call table	Yes
enyelkm	LKM	Kernel text	Yes
override	LKM	Sys call table	Yes
adore-ng-0.56	LKM	Virt file system	Yes
mood-nt	Raw mem. access	Sys call table	Yes
superkit	Raw mem. access	Kernel text	Yes
suckit2priv	Raw mem. access	Kernel text	Yes

Table 1 Linux kernel-level rootkits tested on RootkitLibra.

Our test set of rootkits represent different targets and use variety of attack vectors to subvert the kernel. So, we believe that any other rootkits that are not in the test set but use same attack vectors or target should be detected by RootkitLibra. Table 1 shows the tested rootkits, their attack vectors and targets and the detection status by RootkitLibra.

The tested rootkits showed various behaviors being watched by RootkitLibra and thus were caught when committing those behaviors. The hidden files appear in the VMM-level view but do not appear in the OS-level view. Basically, all these rootkits hide their own source or object files and/or directories mostly during their installation mainly with the objective to persist after system reboot. Some rootkits allow the attacker to select whether they want to persist after system reboot or not and hide files and or directories accordingly.

Of the tested rootkits, *Adore* rootkit replaces 15 system calls, mostly filesystem related ones in the system call table and redirects them to its own version of those system calls. It uses a user-space program called *ava* to hide files. Similarly *Adore-ng* patches itself to the virtual filesystem layer of the kernel and intercept accesses to `/proc` filesystem. *adore-ng* rootkits hid the source directory right after they were installed. Once *adore-ng* is installed and is in kernel, it intercepts all the reads of the filesystem and checks uid and gid. If the uid and gid is set to its *elite* value, the files and directories are hidden. Adore-ng has a configure script which does a `chown` to its *elite* uid and gid value to the source directory and successfully hide it. As shown in the Table 1, both these rootkits use loadable kernel modules (LKM) support in commodity Linux kernels to directly inject the code that performs these malicious activities.

Enyelkm hid its kernel object file in the `/etc` directory right after it got installed. *Enyelkm* tries to persist even after system reboot. It inserts a line of code in `/etc/rc.d/rc.sysinit` to load the hidden rootkit kernel object file in `/etc` directory after system reboot. *Enyelkm* basically injects code in the system call handler and taps each system call to hook and modify the system call outputs of some of the crucial filesystem related system calls such as `getdents, read` and `kill`. *Enyelkm* also uses LKM to inject code directly to kernel text.

Mood-nt, superkit and suckit2priv rootkits are influenced by earlier *suckit* rootkit. They all use Linux's `/dev/kmem` device to inject rootkit code directly into the kernel. *Mood-nt* hijacks 44 system calls which includes almost all of the filesystem related system calls. It created a hidden directory to keep the `init` file that it planned to use while rebooting the system. It also hid the original `/sbin/init` file in the same location after appending its name with its chosen string that was used during file and directory hiding. *suckit2priv* hijacks 45 system calls. These includes all the filesystem related system calls. Similarly, *superkit* hooks 25 system calls that includes almost all the filesystem related system calls. and suckit2priv hid the `/sbin/init` file after renaming it and inserted their own patched init file in the same location.

Override and *Rial* rootkits have lots of similarity in the way they work. They both are use LKM to get into the kernel and both of these rootkits patch number of filesystem related system calls to perform malicious activities. *Override* patches 9 system calls whereas *Rial* patches 5 system calls. They hide files that begin with certain user-defined prefix. The version of these rootkits we found were meant for the educational purpose and it did not hide anything immediately after they were installed. So, RootkitLibra did not detect their presence right after they were installed. But once we deliberately chose to hide files or directories, RootkitLibra caught the behavior.

In general, any malware which persists across system reboot cannot go undetected with RootkitLibra as it must hide certain files in the process to persist after reboot. But, if the rootkit does not show any behavior monitored by RootkitLibra right after it is installed, then RootkitLibra will not detect its presence until the rootkit hides some file or directory or changes some file attributes but hides those changes from the administrator. But, as we all know, the fundamen-

Benchmark	w/o RootkitLibra	w/- RootkitLibra	%
PostMark	802.5	813	1.31%
find	27.93	30.03	7.52%

Table 2 Performance of macro benchmark.

tal behavior of a rootkit and similar malware is to hide other malware and itself. So, if the malware does not hide itself or any other malware then it is caught anyway.

5.2 Performance Overhead

All of the performance overhead experiment uses Xen 3.0.2-2 as VMM and linux 2.6.16 as the monitored system. The monitored systems' filesystem uses NFS protocol version 3. The complete filesystem resides in NFS server. The test system is a Pentium D (3.00GHz) PC with 1GB RAM and a SATA hard disk drive. The Xen control domain(Dom0) is configured with 873MB of RAM while the monitored OS(DomU) is configured with 128MB RAM. Similarly, machine that hosted NFS Server is Pentium 4 (3.4GHz) PC with with 1GB RAM and SATA hard disk drive. Both NFS server and client are on the same 1.0Gbps network switch.

For macro benchmark test, we used Postmark[14] benchmark and `find` utility program. Postmark was basically designed to measure the performance of a file system used for electronic mail, netnews and web based services. It creates a large number of randomly sized files and performs a specified number of transactions on those files. Each transaction has two sub-transactions viz. create or delete and read or append. The configuration we chose consisted 100,000 transactions across 20,000 files and 1,000 subdirectories with equal biases for transaction types. The size of the randomly created files were between 512B and 16KB.

Table 2 shows that RootkitLibra adds a very minimal overhead of 1.31% to the system runtime when tested with *Postmark* filesystem benchmark. Similarly, it also shows the result of `find` command. The reason for a slightly high overhead during `find` is because when we use `find` command based on inode of a file, it traverses each and every file, directory and device in the filesystem and retrieves various filesystem data/metadata during its runtime. RootkitLibra extensively uses those data and hence the higher overhead. This overhead possibly represents the worst-case runtime overhead due to RootkitLibra.

6. Discussion

Despite being very robust, tamper proof and effective against kernel-level rootkits our system has some limitations. The limitations arise due to the following reasons.

Transient malware that does not alter filesystem information Since RootkitLibra is a behavior based malware detection system, it relies on a fundamental behavior of a typical rootkit or any other malware whereby the malware attempts to hide various files, directories or make changes to the attributes of different files in a filesystem and hide those changes from the administrators. But, there can be a malware which is transient in nature and does not aim to persist and does not show any behavior monitored by RootkitLibra. Such malware evades the detection process of our system.

VM Detection There is every chance that a clever attacker can detect the presence of VMM layer beneath the OS[17),21),22]. Running an OS in a virtualized environment leaves many tiny footprints such as the prolonged time for I/O operations, device access, certain virtualized instructions compared to the similar timings in non-virtualized environments[10]. If an attacker uses these techniques to detect the virtualized environment and thus change its behavior, RootkitLibra may not detect the malware activity. But there have been efforts to counteract the VM fingerprinting too[18].

Virtualization based rootkits King et al.[16] first proposed the concept of VM-based rootkit which gets installed beneath the host OS. Similarly, there have been efforts to build hardware virtualization based rootkit such as HVM-rootkit[19], Blue Pill[23] and Vitriol[26]. These rootkits work in VMM layer where RootkitLibra resides. In effect, these rootkits can compromise RootkitLibra and render it ineffective.

Advanced rootkits capable of changing System Call Interface RootkitLibra relies on the knowledge of system call interface of the host it is monitoring. System call interface is a pretty stable interface which is hardly changed from version to version of OS. Rather than being changed, system call interface is generally extended in a newer OS version. This reliance on stable interface like

system call interface adds to the portability of RootkitLibra. But, if there comes a powerful rootkit which is capable of changing the system call interface, RootkitLibra may not work correctly. We have not encountered any rootkit which has this capability.

7. Related Work

There have been a number of past work towards an efficient and robust IDS that use virtualization[8),9),12)]. In this section we would like to analyze some of the previous work and discuss some of the similarities and dissimilarities with our work.

VMI[9)] started the concept of placing the IDS outside the VM it is monitoring for isolation and robustness. Its *livewire* system basically analyzes the low-level VM states such as memory pages and disk blocks of a particular VM from outside the VM. Similarly, **Storage-based Intrusion Detection System**[20)] exploits the isolation obtained from the use of a file server such as NFS Server. While VMI uses various policies that examine the low-level VM states to detect malware, Storage-based IDS uses rules to monitor malware-like activities in remote storage area. These policy/rule based systems have rules which often mistake legal filesystem activity as malware-like activity and hence result in a number of false-positives. But RootkitLibra uses behavior-based approach and cross-view technique both of which have numerous advantages over traditional approaches. Moreover, RootkitLibra uses some semantic gap bridging techniques to get high-level states of VM in VMM and also do not have any false positives. Also, Storage-based IDS requires a separate trusted channel between the storage device and a trusted admin console but our system does not need any extra channels and relies solely on the trusted isolation provided by VMM.

VMWatcher[11)] elevates the VM Introspection methodology to provide "out-of-the-Box" malware detection capability. VMWatcher provides the capability to run off-the-shelf anti-malware tools outside the host OS by bridging the semantic gap. The semantic view reconstruction process used by VMWatcher uses substantial OS specific knowledge such as the knowledge of process table. This dependency on a specific OS version requires it to have different semantic view reconstruction process for different OS versions. This loses its portability. But RootkitLibra does not rely on any version specific details of an OS but rather makes use of the knowledge of stable system interface such as system call interface and hence is more portable.

Copilot[13)] monitors the presence of kernel rootkits by periodically scanning the kernel memory area. It runs the monitoring system in a separate PCI card thereby keeping a strong isolation between the detection system and malware. While this system presents a robust detection model the necessity of a separate PCI card and a firmware into it makes it very rigid and inflexible. Also, because of the strict dependence on objects like kernel modules of some specific version of OS into the firmware, this approach cannot be easily ported to different versions of OS. On the other hand, our approach provides a strong isolation to the rootkit detection system from a possible malware attack due to the use of virtualization technology. In addition to it, our system does not depend on any particular versions of OS or VMM.

Strider Ghostbuster[5)] pioneered a cross-view approach that compares the user level view with the kernel level view of the same system as well as compares the "internal view" with the "external view" in order to detect hidden files and processes. If the two views present any inconsistencies the system raises a flag for malware detection. The major disadvantage of this approach is that the whole detection process is not swift and needs the entire disk scan along with the reboot of the system from a trusted OS. On the other hand, RootkitLibra is very efficient regarding the overhead incurred and also runs live with the host OS without the need of any system reboot for any comparison or detection.

Along with these virtualization based intrusion detection systems, there have been lots of efforts from different areas on developing various host-based IDS[1),4),15)] and network-based IDS[3)]. These host-based systems do not have the problem of overcoming semantic gap but have a strong challenge to remain isolated before being subverted by advanced malware like kernel-level rootkits. Network-based detection system on the other hand have relatively higher isolation from any possible malware in the system compared to their host-based counterparts but lack the access to important system states. RootkitLibra by virtue of virtualization maintains a strong isolation from malware at the same time acquires enough semantic view of the host to perform its job effectively.

8. Conclusion

As the malware technology develop further and target various kernel entities, it is of utmost importance to guard the kernel and the integrity of filesystem. In this paper, we have presented RootkitLibra, a VMM-based simple and efficient filesystem integrity checker and rootkit detector capable of detecting the stealthy and elusive kernel-level rootkits. RootkitLibra, by virtue of its location in VMM is tamper-proof and it acquires OS view of files/directories and related information in VMM through various semantic gap bridging techniques in order to effectively detect the presence of malware. Our experiments with 8 real world rootkits prove its effectiveness and further overhead experiments prove its efficiency.

References

1) Chkrootkit. http://www.chkrootkit.org/.
2) Packet Storm Security. http://packetstormsecurity.org/UNIX/penetration/rootkits/.
3) Snort. http://www.snort.org/.
4) The Samhain File Integrity/Host-based Intrusion Detection System. http://www.la-samhna.de/samhain/.
5) D.Beck, B.Vo, and C.Verbowski. Detecting Stealth Software with Strider GhostBuster. In *Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN '05)*, pages 368–377. IEEE Computer Society, June 2005.
6) D.P. Bovet and M.Cesati. Understanding the Linux Kernel (3rd Edition), 2006.
7) S.Cesare. SysCall Redirection Without Modifying the SysCall Table, 1999. http://vx.netlux.org/lib/vsc05.html.
8) G.W. Dunlap, S.T. King, S.Cinar, M.A. Basrai, and P.M. Chen. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replaying. In *Proceedings of the 2002 USENIX Symposium on Operating Systems Design and Implementation (OSDI '02)*, pages 211–224. ACM, December 2002.
9) T.Garfinkel and M.Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *Proceedings of the 2003 Network and Distributed System Security Symposium (NDSS '03)*, February 2003.
10) R.P. Goldberg. Survey of Virtual Machine Research. *IEEE Computer Magazine*, 7:34–45, June 1974.
11) X.Jiang, X.Wang, and D.Xu. Stealthy Malware Detection Through VMM-Based "Out-of-the-Box" Semantic View Reconstruction. In *Proceedings of the 2007 Computer and Communications Security (CCS '07)*, October 2007.
12) A.Joshi, S.T. King, G.W. Dunlap, and P.M. Chen. Detecting Past and Present Intrusions throught Vulnerability-specific Predicates. In *Proceedings of the 2005 Symposium on Operating Systems Principles (SOSP '05)*, pages 91–104. ACM, October 2005.
13) N.L.Petroni Jr., T.Fraser, J.Molina, and W.A. Arbaugh. Copilot - a Coprocessor-based Kernel Runtime Integrity Monitor. In *Proceedings of the 13th USENIX Security Symposium*, pages 13–23, August 2004.
14) J.Katcher. Postmark: A new file system benchmark. Technical report tr3022, Network Appliance, October 1997.
15) G.H. Kim and E.H. Spafford. The design and implementation of tripwire: A file system integrity checker. In *ACM Conference on Computer and Communications Security*, pages 18–29, 1994.
16) S.T. King, P.M. Chen, Y.M. Wang, C.Verbowski, H.J. Wang, and J.R. Lorch. Subvirt: Implementing Malware with Virtual Machines. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P '06)*, pages 314–327. IEEE Computer Society, May 2006.
17) T.Klein and Scooby Doo. VMWare Fingerprinting Suite, 2003. http://www.trapkit.de/research/vmm/scoopydoo/index.html.
18) T. Liston and E. Skoudis. Thwarting Virtual Machine Detection, August 2006. http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf.
19) M.Myers and S.Youndt. An Introduction to Hardware-Assisted Virtual Machine (HVM) Rootkits, August 2007. http://crucialsecurity.com.
20) A.G. Pennington, J.D. Strunk, J.L. Griffin, C.A.N. Soules, G.R. Goodson, and G.R. Ganger. Storage-based Intrusion Detection: Watching Storage Activity for Suspicious Behavior. In *Proceedings of the 12th Usenix Security Symposium*, pages 137–152, August 2003.
21) D.Quist and V.Smith. Detecting the Presence of Virtual Machine using Local Data Table. http://www.offensivecomputing.net.
22) J.Rutkowska. Red Pill: Detect VMM using (almost) One CPU Instruction, November 2004. http://invisiblethings.org/papers/redpill.html.
23) J.Rutkowska. Subverting Vista Kernel for Fun and Profit [Online]. In *Black Hat 2006*, August 2006.
24) A.C. Arpaci-Dusseau S.T.Jones and R.H. Arpaci-Dusseau. Antfarm: Tracking Processes in a Virtual Machine Environment. In *Proceedings of the 2006 USENIX Annual Technical Conference)*, pages 1–14, June 2006.
25) A.Shah. Analysis of Rootkits: Attack Approaches and Detection Mechanism. http://www-static.cc.gatech.edu/~salkesh/files/RootkitsReport.pdf.
26) D.A.D. Zovi. Hardware Virtualization Rootkits. [Online]. In *Black Hat 2006*, August 2006.