Rootkit Detection on Virtual Machines through Deep Information Extraction at Hypervisor-level

Xiongwei Xie
Department of SIS
UNC Charlotte
Charlotte, NC 28223
Email: xxie2@uncc.edu

Weichao Wang
Department of SIS
UNC Charlotte
Charlotte, NC 28223
Email: weichaowang@uncc.edu

***Abstract*—As a special type of stealth attacks, a rootkit hides its existence from malware detection and maintains continued privileged access to a computer system. The proliferation of virtualization creates a new technique for the detection of such attacks. In this paper, we propose to design a rootkit detection mechanism for virtual machines through deep information extracting and reconstruction at the hypervisor level. Through accessing the important components of a VM such as the kernel symbol table, the hypervisor can reconstruct the VM's execution states and learn the essential information such as the running processes, active network connections, and opened files. Through cross-verification among the different components of the reconstructed execution states of the VM, we can detect both the hidden information and the anomaly connections among them. We implement our approach in Xen 4.1 with Linux VMs. Our experiments show that the hypervisor can efficiently reconstruct the semantic view of a VM's memory and identify the rootkits. Since the hypervisor accesses only the high level data structures, it has very limited impacts on the performance of VM.**

I. Introduction

Computer systems face the threats from many kinds of stealth attacks such as rootkits [1]. A rootkit is a stealthy type of software, often malicious, designed to hide the existence of certain processes or programs from normal methods of detection and enable continued privileged access to a computer [2]. Once attackers have obtained root or administrator access to a system, they will install rootkits to hide the evidence so that system administrators cannot detect them. In addition to stealing sensitive information, attackers also use rootkits to create backdoors for subsequent attacks.

Rootkits are difficult to detect since a rootkit tries to hide its existence from anti-malware programs. Existing approaches to rootkit detection can be classified into three groups. In the first group, researchers analyze and characterize the behaviors of the rootkits [3], [4], [5]. For example, HookFinder [3] provides valuable insights and details about the underlying hooking mechanisms that are used by attackers. K-Tracer [4] and Panorama [5] are automatic tools that can efficiently analyze the data access and propagation paths and the manipulation behaviors of different software. In the second group, researchers try to detect the rootkits through certain symptoms that are exhibited by the intrusion. For example, SBCFI (state-based control-flow integrity) [6] monitors kernel integrity of the operating system to detect malicious changes. Copilot [7] implements a similar approach in coprocessor platforms. In the third group, approaches are designed to prevent rootkits from changing the OS kernel. In [8], the authors present a technique that uses static analysis to identify instruction sequences of malicious activities based on their signatures.

The host-based rootkit detection mechanisms have their limitations. For example, some rootkits such as Agobot variant [9] can detect and remove more than 105 types of anti-malware programs in the victim machine. The emergence of cloud computing opens a new horizon for solving this problem. In a virtualized environment, the hypervisor can monitor the behaviors of the virtual machines. While a rootkit may be able to fool the guest OS, it will be very difficult to hide a malicious process from the hypervisor. Several security systems have been developed for rootkit detection in virtual machines. For example, VMwatcher [10] uses the general virtual machine introspection (VMI) [11] methodology in a non-intrusive manner to inspect the low-level VM states. UCON (usage control model) [12] is an event-based logic model. It maintains the lowest level accesses to the system and ensures that such accesses cannot be compromised by internal processes of a VM.

Existing hypervisor-based rootkit detection mechanisms use information from different modules of a VM as individual components to conduct malware detection. Since the data is not cross-verified among different modules, some malware could have escaped detection. For example, VMwatcher [10] compares the process names at the VM level and hypervisor level to identify any hidden programs. An attacker can change the process' name to a frequently-used text editor to avoid detection. However, when we cross-examine the process table with opened files, we may detect that the text editor has opened a TCP connection and turned on the microphone. The mismatch of information among different modules will allow us to detect the rootkit that hides deeper in the system.

Moreover, several software packages that are distributed by famous companies have been found to contain malicious components. For example, in 2005, Sony BMG published CDs with copy protection and digital rights management software called Extended Copy Protection. The software silently installed a rootkit that would limit the user's ability to access the CD and change system configuration without a user's

978-1-4799-0895-0/13/$31.00 ©2013 IEEE

permission [13]. Under this condition, some existing rootkit detection software may also fail.

In this paper, we propose a rootkit detection mechanism based on deep information extraction and cross-verification at the hypervisor level. Since the hypervisor sees only the raw memory pages of a virtual machine, we need to first reconstruct the semantic view of a virtual machine's memory in order to recover its execution states. The recovered information includes processes, network connections, kernel-level modules, and opened files. After reconstructing the semantic view of a VM's memory, we examine the execution states that are directly obtained from the VM and those reconstructed by the hypervisor. Through cross-verification among different modules of the two views, we can find discrepancy between them and identify the hidden malware in the guest OS.

While the basic idea is straightforward, we must overcome two challenges to turn it into a practical approach. First, there is a "semantic gap" [14] between the memory viewed by the virtual machine and that by the hypervisor. In hypervisor, we see only the raw memory pages, registers, and disk blocks. Therefore, we must establish a native view of the VM's memory just like we are in the VM. The second challenge that we face is to cross-verify different modules of the reconstructed memory view and identify any mismatch in the information. In this preliminary version of research, we define a static table that links frequently used applications to the file types and network connections that they can operate on. A more intelligent approach will be designed in future work.

Compared to existing rootkit detection mechanisms in virtualization environments, the proposed approach has the following advantages. First and most importantly, we use the reconstructed information from different modules of the VM as an integrated, cohesive system for rootkit detection. This provides us a stronger detection capability than existing approaches. Second, our rootkit detection mechanism uses non-intrusive introspection of VMs. Therefore, it incurs very limited overhead in the VMs. Last but not least, we implement the proposed approach in XEN and show that it has very small performance impacts on the virtualization environment.

The remainder of this paper is organized as follows. In Section II we describe the details of the proposed approach. We discuss the reconstruction of the semantic view of a VM's memory and the cross-verification procedures. In Section III we present the implementation of the rootkit detection mechanism and assess the performance impacts when XEN Hypervisor is used. Section IV discusses the communication component of the detection procedures and analyzes its safety. Finally, Section V concludes the paper.

II. The Proposed Approach

A. System assumptions and design goals

In the investigated scenario, we assume that an attacker has acquired the administrator privilege in the target VM and she/he can install malware in the system. The rootkit embedded by the attacker can modify the returned results to the auditing programs on the VM (e.g., ps, netstat or lsof on the guest OS) to hide the intrusion. The attacker may also leave a backdoor in the system for subsequent attacks. However, similar to the approaches in [15], [16], we assume that the attacker cannot compromise the hypervisor.

Our investigation has the following design goals. First, the proposed approach should be transparent to end users. It can accurately extract and recover the VM's execution states without any help from the virtual machine. Users will be notified only when security alarms are raised. Second, we need to minimize the performance impacts of the proposed approach on the target virtual machine. This requirement will help us avoid difficulty in future adoption of the mechanism. Third, the proposed approach must be hypervisor independent. The design should support VMM in both full virtualization (e.g. KVM [17] and VMware [18]) and paravirtualization (e.g., XEN [19]). This property will allow more users to benefit from the approach.

B. Memory reconstruction

The first step of the proposed approach is memory reconstruction of the virtual machines at the hypervisor level. Through memory reconstruction, we can extract the high level semantic information of the VM. To accomplish this procedure, we need to locate the static data entries that are essential for the kernel and the boot-up procedures. Since many of these entries are accessed frequently, their addresses are determined at the compilation time and can be found from the kernel symbol table (i.e., $System.map$ in Linux), which can be viewed as a look-up table between symbol names and their addresses in memory.

In our approach, we first reconstruct the disk image of the virtual machine on the hard drive in order to get access to the kernel symbol table (i.e., $/boot/System.map-\$(uname-r)$ in Linux). Once we know what the file system is and how the files and directories are organized in the virtual disk, we can reconstruct the semantic view of the virtual disk from the raw virtual disk. For example, it is easy to reconstruct the raw virtual disk of a Linux VM because of its open-source kernel structure. The reconstruction results will then allow us to determine the memory addresses of the static variables in the kernel symbol table.

To reconstruct the semantic view of a VM's memory, we need to translate the memory address in a virtual machine to the corresponding physical address in the host machine. Through the reconstruction of the virtual disk, we already know the addresses of many static variables. For example, $init_task$ is a statically declared task that is at the head of the process list in Linux. Starting from this position, we can go through the complete process list and get the information of all processes using the offsets of the entries such as pid, process name, and the pointer to the next process.

To better explain the memory reconstruction procedure, below we use an example of a 32-bit Linux guest OS to illustrate the information extraction operations. The procedure is shown in Figure 1. In this example, the user space occupies the bottom 3GB memory while the kernel occupies the top

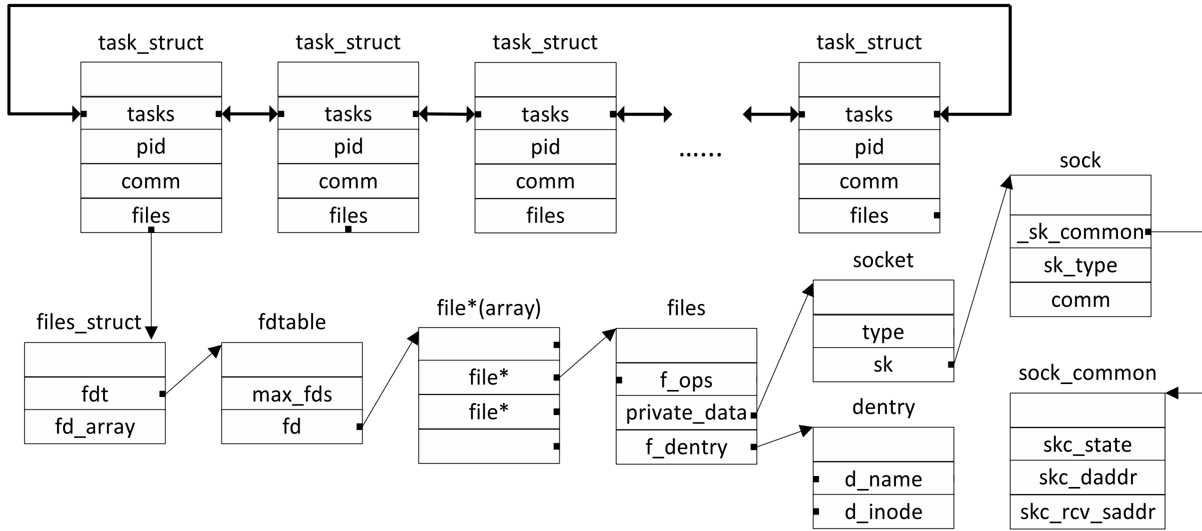

Fig. 1. Semantic view of a virtual machine's memory through memory reconstruction.

1GB. From the kernel symbol table, we know that the address of $init_task$ is $0x81c0d020$. The offset of tasks (struct $list_head$) is $0x240$. Therefore, the starting address of the first process is *($init_task$+$offset_tasks$) - $offset_tasks$. Using the similar technique, we can reconstruct a number of other important data structures such as files and sockets. We have used several functions such as $vmi_read_addr_va$ and $vmi_read_32_va$ to translate the virtual addresses to corresponding physical addresses. While we use memory reconstruction of Linux as an example in this paper, the same technique can be applied to other operating systems such as Windows through investigation of the Windows Symbol packages.

C. Rootkits detection based on cross-verification

As described in Section I, it is not sufficient to only examine whether or not the execution states at the VM level and those at the hypervisor level match with each other. In this part, we design a rootkit detection mechanism that explores deep information about the VM, especially the relationships among the running processes, active network connections, and opened files to detect the anomalies caused by malware. The procedure is illustrated in Figure 2.

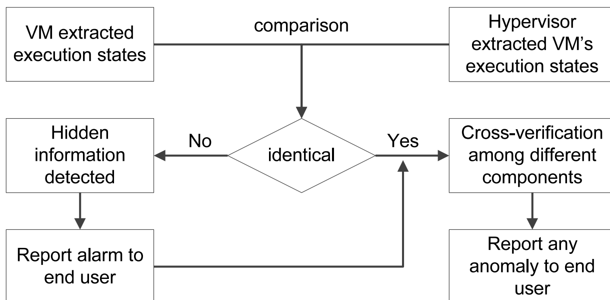

Fig. 2. The proposed rootkit detection procedure.

In our approach, we use a socket for the communication between the hypervisor and the VM. We have implemented a module in each side to support the information exchange and rootkit detection operations. While the detection procedures can be initiated by either side, below we describe one scenario in detail. When a VM suspects that it is under attack, it can send a message to the hypervisor to start the detection procedures. Then both sides will acquire the execution states of the VM with their own methods. Once the extraction operations are accomplished, the VM will send the results to the hypervisor so that it can compare the information. If any discrepancy is detected, the hypervisor can raise an alarm of the hidden information. The hypervisor will then cross-examine different components of the extracted information. In our preliminary implementation, we study the relationships among the processes, network connections, and opened files. Using the guest OS Ubuntu (64bit) with 3.5.0-23-generic kernel, we have manually generated a static table for the processes of frequently used applications. In this table, we have defined the file types that a process can open, whether or not this process can be associated with network connections, and any special properties of the connections (such as the number of concurrent connections, the protocols, and the port numbers). The hypervisor will cross-examine different components of the extracted execution states against this table. If any violation is detected, an alarm will be sent to the end user. In the next section, we will present a concrete example to show the effectiveness of the proposed approach.

III. Implementation and Experimental Results

A. Experiment Setup

To evaluate the detection capabilities of the proposed approach and assess its impacts on the system performance, we conduct two groups of experiments on the hypervisor Xen version 4.1 with the library libvir 0.9.8. In the first group, we test our approach on Paravirtualization (PV) over Xen. The host OS is Ubuntu Desktop 12.04 LTS (64bit). The PV guest OS is Ubuntu (Precise 64bit) with 3.5.0-23-generic kernel. In

the second group, we test our proposed approach on Hardware Virtual Machine (HVM) guest. The host OS is Ubuntu Server 12.04 LTS (64bit). The HVM guest OS is Ubuntu Desktop 10.04 LTS (32bit) with 2.6.32-24-generic kernel.

While the virtual machines can use different types of guest OS, below we use Linux as an example to show the information extraction procedure. As shown in Figure 1, in Linux for any opened file or socket we can access the data structure $file$ to get its information. Through comparing the field f_ops to the value of $socket_file_ops$ in the Kernel Symbol Table, we can determine whether the handle is for a file or a socket connection. If the handle actually points to a file, we can use the field d_name in the structure $dentry$ to get the file's name. Otherwise, we can use the $socket$ structure to get the information about the network connection.

B. Experiment Results

View comparison-based rootkit detection discovers malware through finding information mismatch between the hypervisor and the virtual machine. The discrepancy could be caused by either information hidden by the malware, or the anomaly links among the processes, files, or network connections. Below we use a concrete example to illustrate the detection capability of the proposed approach. Here we use an advanced Linux kernel rootkit KBeast as the investigation target. KBeast can hide loadable kernel modules, processes (ps, top, lsof, pstree), socket and connections (netstat, lsof), and anti-kill processes from the infected OS. Furthermore, it leaves a hidden backdoor open for subsequent attacks. KBeast also allows the attackers to change its process name so that it looks like a benign application. It is so stealthy and elusive that many anti-malware packages such as chkrootkit and rkhunter cannot detect it. Figure 3 shows the screenshot of an infected VM in which the KBeast runs and hides a process with the PID 1788. In Figure 3, the background GUI screen on the right side shows the inside view of the VM while the foreground screen on the left side shows the reconstructed semantic view of the memory of the same virtual machine.

In the figure, we can observe that the rootkit KBeast is built successfully and it is running with PID = 1788, which is hidden from the virtual machine. The attacker has also changed the process' name to 'pdf-reader' so that looking at only the process table is not sufficient for its detection. The malware opens a backdoor for remote access through telnet. In the virtual machine, we cannot detect any anomaly through ps or netstat. The reconstructed view of the virtual machine at the hypervisor level is shown on the left side of the figure. In the left bottom of the figure, we can see that the proposed approach reveals a running process with PID=1788 called *pdf-reader*. If we look at only this information, we may assume that the virtual machine has the application such as Acroreader running. However, when we link the processes to opened files, we find that this process has opened four files/connections. The first TCP connection is the telnet backdoor and its state is CLOSE_WAIT. The state of the second TCP connection is LISTEN. It is the backdoor that KBeast opens. Although many applications these days depend on network connections to achieve software updates, it is still a very suspicious activity when the PDF reader holds an open connection and waits for external requests. The proposed approach detects this anomaly and reports it to the VM.

In addition to the experiments described above, we have conducted another group of experiments to test the proposed approach upon a PV virtual machine. Here the rootkit replaces the system auditing programs (including lsof and netstat) with some malicious interfaces so that it can control the returned contents. Our experiments show that the proposed approach can also reveal the hidden information through memory reconstruction, including the opened files, Socket connections, and the IP addresses of the communication parties. The experiments show that our approach is not restricted by the type of virtualization.

C. Overhead and Performance Analysis

To protect a VM from rootkit infection, we need to execute the proposed approach at the hypervisor level periodically. Since we need to freeze the VM during memory reconstruction, we must study the relationship between the detection frequency and its impacts on the system performance. We conduct two sets of experiments to assess the impacts.

The first experiment tries to measure the memory reconstruction time at the hypervisor level. Since each memory reconstruction consumes a very short period of time, we measure the accumulative delay of 5000 reconstructions. In order to simulate the real working environment, after each memory reconstruction the host OS will sleep for one second so that the VM will get the CPU back. Our measurement shows that the average execution time of memory reconstruction is about $20ms$ regardless of the size of the allocated memory of the VM since we access only its high level data structures.

In the second group of experiments, we try to evaluate the impacts of the proposed rootkit detection mechanism on the performance of the guest VM. We choose three applications in the guest VM as benchmarks and measure their execution time when the rootkit detection frequency is changed. As shown in Figure 4, the $make$ benchmark compiles the kernel of Linux version 3.10 and incurs intensive CPU and I/O workload. The $gzip$ benchmark compresses a large file and demands more resources from the CPU. Lastly, the $find$ benchmark tries to locate a specific file on the hard-drive and incurs intensive disk and file system accesses.

From the figure we find that when we increase the interval between the rootkit detection, its impacts on the VM performance are decreasing. When the interval is equal to 125 seconds, the applications are almost not affected at all. Even when we execute the proposed approach every second, the increase in application execution time is less than 2.5%. Based on the results, we conclude that our proposed approach introduces very low overhead.

IV. DISCUSSION

While there are many different ways for hypervisors to communicate with guest VMs, in our implementation we use

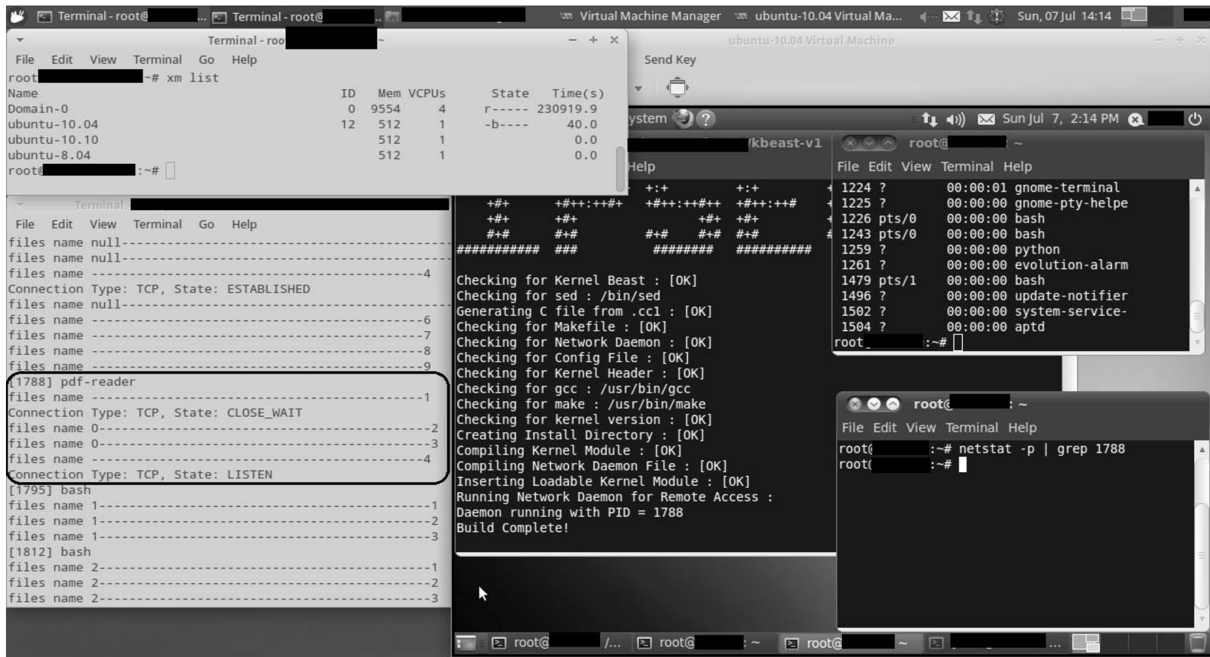

Fig. 3. The detection of KBeast in Xen through cross-verification.

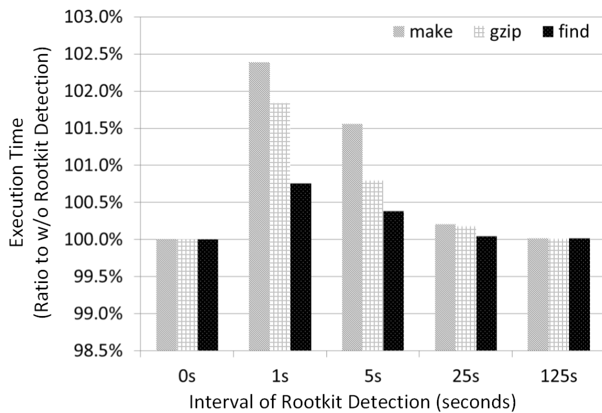

Fig. 4. Relationship between rootkit detection frequency and its impacts on system performance.

sockets to allow the two parties to exchange information. We establish a socket in the hypervisor that listens to the rootkit detection requests from VMs. When a request is received, the hypervisor will temporarily freeze the VM and access its memory. After exchanging the extracted information, the hypervisor will send the malware detection results back to the VM. To protect authenticity and integrity of the data, the hypervisor can digitally sign the hash result of the exchanged information. The two parties do not need accurate synchronization to look at the same snapshot of the memory since the hypervisor does not solely depend on the information provided by the VM to detect suspicious activities.

Since the proposed approach tries to detect malicious software in computer systems, we must carefully assess its safety. Below we analyze two scenarios. In the first scenario, the rootkit actually controls the communication channels between the VM and the hypervisor. Under this condition, the rootkit can modify the data that is transmitted to the hypervisor so that the two memory views are identical. However, such changes will not prevent its detection since the cross-verification procedure at the hypervisor level can still identify the anomalies. Another operation that the rootkit can take is to totally disable the communication with the hypervisor. Under this condition, the hypervisor cannot effectively notify the end user even if the malware is detected. To solve this problem, a notification method that demands user interactions can be adopted to differentiate an automatic software from a real user.

In the second scenario we are facing attackers with more advanced skills. Here the malware can intentionally modify the significant values of the system in the Kernel Symbol Table. Under this condition, we cannot reconstruct a true semantic view of the VM's memory at the hypervisor level. The attacker can also alternate other components of the kernel to fabricate a false view so that the information from the VM and that from the hypervisor match to each other and no anomaly can be identified. Fortunately, changes to OS kernels can be detected through attestation of the integrity of the images [20].

V. Conclusion

In this paper, we propose a new rootkit detection mechanism for virtual machines through deep information extraction and reconstruction at the hypervisor level. The hypervisor will first rebuild the semantic view of the VM's memory. Through cross-verification among different components of the reconstructed view, the hypervisor can detect hidden information and mismatch among different active modules in the VM. Our experiment results show that the proposed approach is practical and effective in rootkit detection. Furthermore, the

performance overhead is very low since we access only the high level data structures of the VM.

Immediate extensions to our approach consist of the following aspects. First, we plan to experiment our approach with Windows VMs so that we can evaluate its practicability in other environments. Second, we will introduce intelligence into the construction of and anomaly detection in the linkage table among different modules of the VM. Finally, we plan to extend our approach to other hypervisors so that more end users can benefit from our research.

REFERENCES

[1] M. Jakobsson and Z. Ramzan, "Evolution of rootkits," in *Crimeware: Understanding New Attacks and Defenses*. Addison-Wesley, 2008.

[2] McAfee, "Rootkits, part 1 of 3, the growing threat," 2006.

[3] H. Yin, Z. Liang, and D. Song, "Hookfinder: Identifying and understanding malware hooking behaviors," in *NDSS*, 2008.

[4] A. Lanzi, M. Sharif, and W. Lee, "K-tracer: A system for extracting kernel malware behavior," in *NDSS*, 2009.

[5] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Krida, "Panorama: Capturing system-wide information flow for malware detection and analysis," in *ACM CCS*, 2007, pp. 116–127.

[6] N. L. Petroni Jr. and M. Hicks, "Automated detection of persistent kernel control-flow attacks," in *ACM CCS*, 2007, pp. 103–115.

[7] N. L. Petroni Jr., T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot: a coprocessor-based kernel runtime integrity monitor," in *USENIX Security Symposium*, 2004.

[8] C. Kruegel, W. Robertson, and G. Vigna, "Detecting kernel-level rootkits through binary analysis," in *Annual Computer Security Applications Conference*, 2004.

[9] "Agobot," http://www.f-secure.com/v-descs/agobot.shtml, 2012.

[10] X. Jiang, X. Wang, and D. Xu, "Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction," in *ACM CCS*, 2007, pp. 128–138.

[11] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *NDSS*, 2003.

[12] M. Xu, X. Jiang, R. Sandhu, and X. Zhang, "Towards a vmm-based usage control framework for os kernel integrity protection," in *SACMAT*, 2007, pp. 71–80.

[13] "Xcp.sony.rootkit," http://archive.is/20120921/http://www.ca.com/us/ securityadvisor/pest/pest.aspx?id=453096362.

[14] P. M. Chen and B. D. Noble, "When virtual is better than real," in *Hot Topics in Operating Systems VIII*, 2001.

[15] A. Yu, Y. Qin, and D. Wang, "Obtaining the integrity of your virtual machine in the cloud," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2011, pp. 213–222.

[16] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, and D. Huang, "Nice: Network intrusion detection and countermeasure selection in virtual network systems," *Dependable and Secure Computing, IEEE Transactions on*, vol. 10, no. 4, pp. 198–211, 2013.

[17] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the linux virtual machine monitor," in *Linux Symposium*, 2007, pp. 225–230.

[18] B. Walters, "Vmware virtual platform," *Linux J.*, no. 63es, 1999.

[19] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of ACM SOSP*, 2003, pp. 164–177.

[20] S. Mei, Z. Wang, Y. Cheng, J. Rena, J. Wu, and J. Zhou, "Trusted bytecode virtual machine module: A novel method for dynamic remote attestation in cloud computing," *International Journal of Computational Intelligence Systems*, vol. 5, no. 5, pp. 924–932, 2012.