VMI Based Automated Real-Time Malware Detector for Virtualized Cloud Environment

M.A. Ajay Kumara(✉) and C.D. Jaidhar

Department of Information Technology,
National Institute of Technology Karnataka, Surathkal, India
{ajayit13f01,jaidharcd}@nitk.edu.in

Abstract. The Virtual Machine Introspection (VMI) has evolved as a promising future security solution to performs an indirect investigation of the untrustworthy Guest Virtual Machine (GVM) in real-time by operating at the hypervisor in a virtualized cloud environment. The existing VMI techniques are not intelligent enough to read precisely the manipulated semantic information on their reconstructed high-level semantic view of the live GVM. In this paper, a VMI-based Automated-Internal-External (A-IntExt) system is presented that seamlessly introspects the untrustworthy Windows GVM internal semantic view (i.e. Processes) to detect the hidden, dead, and malicious processes. Further, it checks the detected, hidden as well as running processes (not hidden) as benign or malicious. The prime component of the A-IntExt is the Intelligent Cross-View Analyzer ($ICVA$), which is responsible for detecting hidden-state information from internally and externally gathered state information of the Monitored Virtual Machine (M_{ed-VM}). The A-IntExt is designed, implemented, and evaluated by using publicly available malware and Windows real-world rootkits to measure detection proficiency as well as execution speed. The experimental results demonstrate that A-IntExt is effective in detecting malicious and hidden-state information rapidly with maximum performance overhead of 7.2 %.

Keywords: Virtual Machine Introspection · Hypervisor · Malware · Semantic gap · Cross-view analysis · Rootkits

1 Introduction

The virtualization platform is becoming an attractive target for an adversary due to easy access of Virtual Machines (VMs) through the cloud service provider [1]. The proliferation of sophisticated rootkit or malware could alter the normal behavior of the legitimate GOS by altering the critical kernel data structures [2–4]. The traditional in-host antimalware defense solution is not only inadequate to thwart advanced malware, but it can also be easily removed by sophisticated rootkits or malware. For example, the malicious logic employed by the Agobot variant rootkit is powerful enough to bypass 105 antimalware defensive processes on the victims machine [5]. To detect the stealthy and elusive malware, the VMI

C. Carlet et al. (Eds.): SPACE 2016, LNCS 10076, pp. 281–300, 2016.
DOI: 10.1007/978-3-319-49445-6_16

[6] has emerged as a tamper-resistant and Out-of-the-Box practical solution to enforce transparently the security assurance on the run state of the GVM [5,7,8].

The VMI is able to gather the run-state information of the M_{ed-VM} without the consent or knowledge of the one being monitored, while functioning at the hypervisor or the Virtual Machine Monitor (VMM). However, obtaining meaningful GVM state information such as process list, module list, system calls details, network connections, etc., from the viewable raw bytes of the GVM memory is a challenging task for the VMI and referred to as the semantic gap [9,10]. To tackle this problem, several approaches have evolved over the last few years by considering different constraints of the GOS [11,12]. However, the current challenges of VMI are: (1) It must have higher scalability features to introspect the rich semantic view of the live state of the GVM. To achieve this, it requires tremendous manual effort to build kernel data structure knowledge of large volumes of GOS [13], (2) The VMI solution requires frequent rewriting of the introspection program due to the dynamic and frequent upgrading of the kernel version, and (3) The VMI must be built with a robust introspection technique that would help to reduce the performance overhead and make the VMI automated with little human effort.

On the other hand, many modern families of malware leverages stealth rootkits functionality to conceals itself, and to evade detection system to tamper other critical kernel data structure such as files, directories, sockets, etc. of the GOS [14,15]. The best way of detecting it is by identifying the hidden running processes. This process is the key source of information for any introspection program to spot the existence of the malware. A prior attempt, the VMM-based Lycosid [16] is aimed at detecting and identifying only the hidden processes (HP) of the M_{ed-VM} at the VMM level. However, the current generation of evasive malware may create new malicious processes (not hidden) or attach itself to the existing legitimate running processes. In such cases, Lycosid is inadequate to detect such a malicious malware process. It does not identify the name or binary of the process and it is also inefficient in checking the detected hidden details are malicious or benign. Moreover, the process visible from the hypervisor may be noisy, most likely incorrect, and may lead to a false positive. Another approach, named the Linebacker [17], also uses the cross view analysis to investigate the rootkit running on the GVM. The efficiency of the Linebacker has been demonstrated on the VMware vSphere-based GVM. However, only the Windows GVMs were considered for evaluation.

The significant challenges in detecting the malicious and dead processes, particularly in a virtualized environment are:

– The number of running processes may significantly differ from time-to-time, even if there are no hidden processes at the moment of introspection (while checking inside the VM and viewed from the VMM). This is due to the fact that the number of processes available in the system is not constant and changes too frequently. This is due to the dynamic nature of the process creation. In such cases, it is highly dubious to rely on the introspected data.

– Estimating accurately the number of dead processes and precisely identifying the malicious processes (not hidden) in a timely manner on the run state of M_{ed-VM} is a challenging task.

In this work, the VMI-based A-IntExt system for a virtualized environment is presented. It mainly detects the hidden, dead and malicious processes that are invoked by rootkits or malware by leveraging Intelligent Cross-View Analysis for Process ($ICVA_p$) algorithm between the externally (VMM-level) captured run-state information and the internally (In-VM level) acquired execution-state information of the M_{ed-VM}. Further, it checks detected, hidden as well as running processes (not hidden) as benign or malicious.

The pertinent contributions of the present work are as follows:

1. We have designed, implemented, and evaluated a consistent and real-time A-IntExt system that periodically scrutinizes the state of the M_{ed-VM} to detect hidden, dead, and malicious processes by leveraging an open-source VMI[1] tool, while functioning at the hypervisor.
2. Our novel A-IntExt accurately detects malicious and hidden-state information on the externally reconstructed high-level semantic view of the M_{ed-VM}, and internally gathers state information of the same M_{ed-VM} by adopting its prime $ICVA_p$ algorithm. Further, it checks detected hidden as well as running processes (not-hidden) as benign or malicious by cross-checking with both local malware database and public malware scanner.
3. A mathematical model of the $ICVA_p$ algorithm has been designed, practically implemented, and implanted into the A-IntExt that detects and classifies suspicious activities of the M_{ed-VM}.
4. The other focus of the A-IntExt is to address the time synchronization problem associated with the internally and externally captured $GVMs'$ state information, which impacts on the hidden-state information detection. This issue is tackled by using the Time Interval Threshold (TIT).
5. The robustness of the A-IntExt was evaluated using publicly available Windows rootkits. In addition, malware was also employed in the experimental work to make the evaluation comprehensive. The A-IntExt correctly detected all of the hidden and malicious state information.

The rest of the paper is organized as fallows: Sect. 2 provides background and related work. Section 3 provides detailed overview of proposed A-IntExt system. Section 4 discusses memory state reconstruction. Section 5 presents experiment and results analysis. The performance overhead of A-IntExt described in Sect. 6. Finally discussion and conclusion addressed in Sects. 7 and 8 respectively.

2 Background and Related Work

To address the semantic gap impediment of the VMI, an attempt was made by Virtuno [9] that creates an introspection program automatically to extract

[1] http://libvmi.com/.

meaningful semantic information using the dynamic slicing algorithm based on the low-level data source of the M_{ed-VM}. The main limitation of this technique is not being fully automated and requires minimal human effort. VMST [11] significantly eliminated the limitation of Virtuoso, by enabling an automatic generation of a secure introspection tool with a number of new features and capabilities. The Virtuoso and VMST paid more attention to bridging the semantic gap, but were unable to satisfy the usefulness and practicality of the VMI. Moreover, these techniques have a high overhead. The system call redirection approach [18] proposed to meet the real-world needs of the VMI by significantly improving the practical usefulness of the VMI, and encouraged one inspection program to inspect the different versions of the GVM. The VMI-based open source tools called Xen Access [7] or LibVMI, VMI-PL [19], Vprobe [25], and HYPERSHELL [20] seamlessly address the semantic gap problem by extracting semantic low-level artifacts of the GVM from the hypervisor specific to the memory state introspection.

Hidden process detection: Antfarm [21] is a VMM based approach incorporated at the VMM to track implicitly and exploit the GOS activities. However, it is insufficient in detecting malicious processes, which are invoked for kernel code alteration. Lycosid [16] extends Antfarm as a VMM-based approach for hidden processes detection and identification, based on implicitly obtained process information from the GOS. Moreover, implicitly obtained information within the hypervisor can be noisy. The authors employed a statistical inference and hypothesis technique to address this challenge. Another out-of-VM hypervisor-based approach, namely, process out-grafting [8] focuses on analyzing the individual process of running all of the VMs processes to identify and detect the suspected process in an on-demand way. Patagonix [22] a hypervisor-based system that detects and identifies stealthily executing binaries regardless of the state of the OS kernel. To achieve this it uses knowledge of the hardware architecture.

The Ghostbuster [23], VM watcher [11] and Lycoside [16] commonly uses cross-view analysis technique to detect and identify of any discrepancy between the trusted view and the untrusted view of the M_{ed-VM}. However, comparison of the semantic data is manually achieved in most of these prior work. The main limitation of the VMI-based cross-view comparison is related to the time synchronization problem associated with the internal and external view acquired. In our work, this issue is addressed using TIT. The $ICVA_p$ is intelligent enough to distinguish between hidden, genuine, and dead processes.

3 Overview of the VMI Based Automated Internal-External System

The goal of the VMI-based A-IntExt system is to enable the inspection tool in a trusted Monitoring Virtual Machine (M_{ing-VM}) to investigate the hidden and malicious run state of the untrusted M_{ed-VM}. The overall architectural design

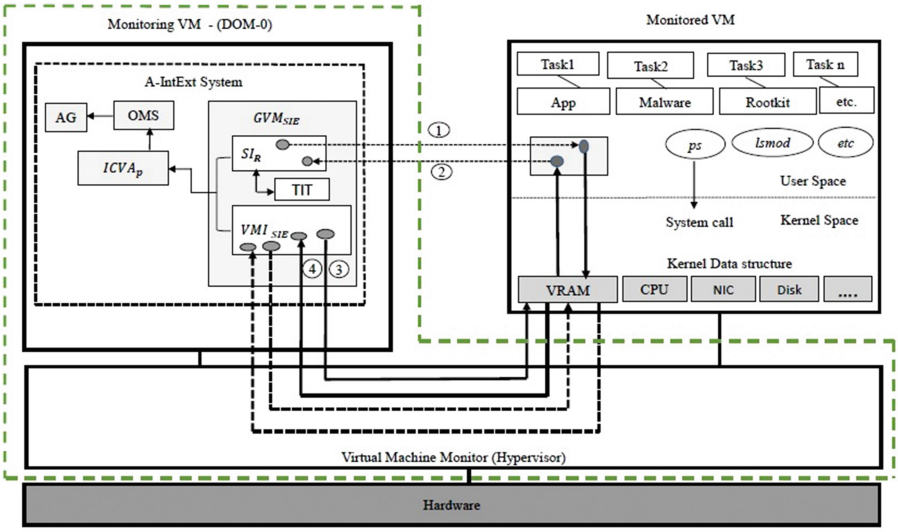


Fig. 1. The proposed VMI based A-IntExt system

of the A-IntExt is shown in Fig. 1. The prime idea is to introduce a hypervisor protected, automated, and independent system to introspect the volatile RAM pages of the M_{ed-VM} from the M_{ing-VM}, and then to identify the hidden execution state by performing an intelligent cross-comparison operation on the internally and externally captured state information. The A-IntExt achieves this goal by using the $ICVA$, which is an integral component. The major components of the A-IntExt are the Guest Virtual Machine State Information Extractor (GVM_{SIE}), $ICVA$, Online Malware Scanner(OMS) and Alert Generator(AG).

3.1 Guest Virtual Machine State Information Extractor

The prime function of the GVM_{SIE} is to extricate the run-state information of the M_{ed-VM}. Its components are: (1) State Information Requester (SI_R), and (2) VMI-based State Information Extractor (VMI_{SIE}). The GVM_{SIE} initiates the process of investigation by signaling the SI_R to send a state-information request to the M_{ed-VM} for currently running processes details. The SI_R makes use of the communication channel established between the M_{ing-VM} and M_{ed-VM} to send a request and to receive a reply. Upon receiving the state-information request (step 1), the M_{ed-VM} acquires the requested data locally, and then sends the results to SI_R after completion of the extraction operation. After receiving the internally gathered state information as a reply from M_{ed-VM} (step 2), the next task of the SI_R is to verify whether a reply arrived within TIT. If the time gap between the state-information request to state-information reply lies within TIT, then GVM_{SIE} immediately acquires the currently running processes of the M_{ed-VM} from the hypervisor to capture the current

PID	Name	Address
[620]	chrome.exe	(struct addr:85be2020)
[2664]	alg.exe	(struct addr:85cd1020)
[2564]	chrome.exe	(struct addr:85bb5020)
[3308]	svchost.exe	(struct addr:85e19020)
[2992]	hxdef073.exe	(struct addr:85eed7b8)

Process name	PID	Session name	Session #	Mem usage
chrome.exe	620	Console	0	101,876 K
alg.exe	2664	Console	0	3,452 K
chrome.exe	2564	Console	0	101,876 K
svchost.exe	3308	Console	0	4,708 K

(a)

PID	Name	Address
[1150]	cmd.exe	(struct addr:85589580)
[2698]	conhost.exe	(struct addr:85cd5898)
[2564]	chrome.exe	(struct addr:85bb5020)
[2589]	csrss.exe	(struct addr:85e19623)
[3355]	Kelihos_dec.exe	(struct addr:85eed7b8)

Process name	PID	Session name	Session #	Mem usage
chrome.exe	1150	Console	0	14,876 K
conhost.exe	2698	Console	0	34,752 K
chrome.exe	2564	Console	0	101,876 K
csrss.exe	2589	Console	0	58,808 K
Kelihos_dec.exe	3355	Console	0	8,5858 K

(b)

Fig. 2. Hidden processes (a) dubious processes (b) details of M_{ed-VM} externally introspected (left side) and internally acquired (right side) by the A-IntExt after rootkit injection on Windows GVM

execution state by directly introspecting the RAM pages of the one being monitored (steps 3 and 4). The SI_R rejects the state-information reply and sends a fresh request whenever the time interval between the state-information request and state-information reply falls outside the TIT.

Figure 2 shows the processes of the M_{ed-VM} captured internally and externally after malware and the rootkit injection, it includes hidden, Dubious processes (DPs[2]) information. This is achieved by the well-built isolation property of the hypervisor guarantees that the state information captured from the M_{ing-VM} is accurate. The procedure followed by the VMI_{SIE} to reconstruct the M_{ed-VM} memory is described in Sect. 4. The GVM_{SIE} sends the gathered state information to the $ICVA_p$ for further analysis.

3.2 Time Interval Threshold

TIT is utilized in the GVM_{SIE} to address the time synchronization problem between the internal and external state-information captures. TIT is the time interval between the state-information request sent to M_{ed-VM} and state-information reply received by the SI_R. Figure 3 demonstrates the TIT used by the GVM_{SIE}. Let T_1 be the date and time at which the state-information request is sent to the M_{ed-VM}, and T_2 be the date and time at which the reply is received by the SI_R from the M_{ed-VM}. Upon receiving the state information, the SI_R

[2] Dubious Processes (DPs) are current state of executable processes it includes both benign and malicious processes (not hidden) on the M_{ing-VM}. Existing hypervisor-based VMI systems are not intelligent enough to detect and identify actual malicious processes that are running or attached to a benign one.

checks the time interval between T_2 and T_1; $T_2 - T_1 > \Delta$T, then the GVM_{SIE} rejects the received state information and resends the state information request, where, ΔT denotes the predefined threshold time. If, $T_2 - T_1 \leq \Delta$T, then, the GVM_{SIE} immediately acquires the execution state of the M_{ed-VM} from the hypervisor.

Assume that the processes $P_1, P_2, P_3 ...., P_N$ are currently being run at M_{ed-VM} and their details are extracted internally during the time interval between T_1 and T_1''. If any process expires or dies after T_1'' and before T_2, such process details do not show up in the state information caught externally by the M_{ing-VM}. As a result, a disparity emerges between the internally and externally captured state information of the M_{ed-VM}.

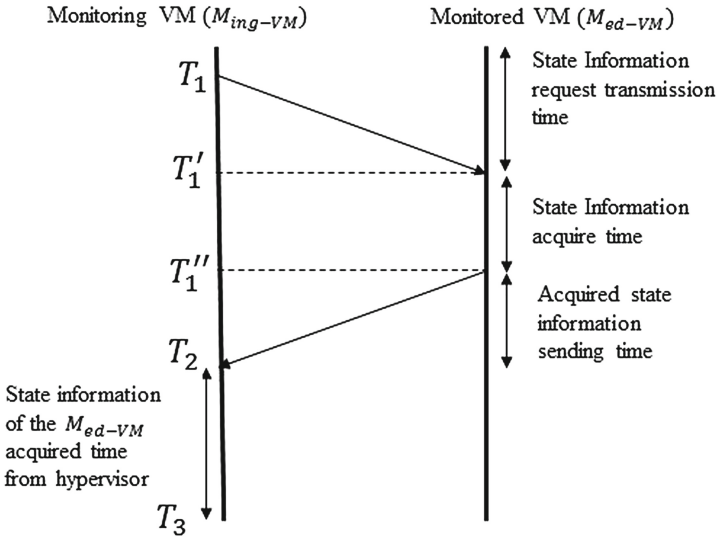


Fig. 3. Time interval threshold used by A-IntExt

The process details appear in the internally captured state information and are absent in the externally captured state information treated as dead processes. In contrast, if a new process P_{N+1} is created between T_1'' and T_3, such process details appear only in the externally captured state information and not in the internally captured state information. As a result, process P_{N+1} is recognized as a hidden process, even though process P_{N+1} is unconcealed. To tackle this issue, first, A-IntExt extracts the entire executable file of the corresponding process, and then, investigates to detect whether any malignant substance is present or not.

3.3 Intelligent Cross-View Analyser

The *ICVA* is an integral component of the A-IntExt and its prime function is to recognize hidden and dead processes of the untrusted M_{ed-VM},

by performing an intelligent cross-examination between the internally and externally acquired execution-state information using $ICVA_P$ algorithm. A-IntExt ascertains the existence of hidden processes by examining Eq. (4); similarly, dead process presence is identified by checking Eq. (6). Further, A-IntExt classifies the introspected processes as hidden and DPs to ascertain whether the detected hidden process and DPs of M_{ed-VM} are benign or malicious by performing a cross-examination with the public OMS, as discussed in Sect. 3.4.

Model for Intelligent Cross-View Analyzer for Processes. The notations used in this section and in Algorithm 1 are depicted in Table 1. The process details captured from the hypervisor (externally) undergo the preprocessing operation, and then stored as $EXT_{ps} = \{PID \parallel PN_1, PID \parallel PN_2, PID \parallel PN_3.....,PID \parallel PN_m\}$ where m = 1, 2, 3,..., and

Table 1. Notations used in the algorithms and their meaning

Symbol	Meaning of the Symbol	Used in
HPC	Hidden Process Count	Algorithms1
DPC	Dead Process Count	
PID	Process Identifier	
PS	Process	
PN	Process Name	
$\parallel$	Concatenation	
PID$\parallel$PN	PID concatenated with PN	
INT_{ps}	Internally Captured Processes	
INT_{psc}	Internally Captured Processes Count	
EXT_{ps}	Externally Captured Processes	
EXT_{psc}	Externally Captured Processes Count	
$EXT_{ps}(PID \parallel PN_m)$	m^{th} PID$\parallel$PN of EXT_{ps}	
$INT_{ps}(PID \parallel PN_n)$	n^{th} PID$\parallel$PN of INT_{ps}	

$PID \parallel PN_m$ represent the m^{th} process. The internally captured process details after the preprocess operation are represented as $INT_{ps} = \{PID \parallel PN_1, PID \parallel PN_2, PID \parallel PN_3.....,PID \parallel PN_n\}$ where n = 1, 2, 3,..., and $PID \parallel PN_n$ represent the n^{th} process. The $ICVA_p$ performs the preprocessing operation to remove unimportant state information and sort the elements of both the EXT_{ps} and INT_{ps} in ascending order, based on the PID.

The total number of EXT_{ps} processes is symbolized as EXT_{psc}

$$EXT_{psc} = \sum_{j=1}^{m} PID \parallel PN_j \tag{1}$$

Algorithm 1. Intelligent Cross View Analyzer for Process ($ICVA_P$)

Input

1: Processes details captured externally from hypervisor stored as EXT_{ps}.
2: Process details captured and sent by the monitored virtual machine (Internally) stored as INT_{ps}.

Output

1: Hidden and dead processes details
2: Hidden Process Count (HPC) and Dead Processes Count (DPC)

1: Pre-process the EXT_{ps} and INT_{ps} such that their elements are in sorted order based on PID
2: Assign HPC=0, DPC=0, p=EXT_{psc}, q=INT_{psc}, n=1, m=1
3: **for all** m such that $1 \leq m \leq p$ **do**
4: **if** $n > q$ **then**
5: Break
6: **else**
7: compare $EXT_{ps}(PID \parallel PN_m)$ with $INT_{ps}(PID \parallel PN_n)$
8: **if** $EXT_{ps}(PID \parallel PN_m) == INT_{ps}(PID \parallel PN_n)$ **then**
9: m=m+1; n=n+1; goto step 4;
10: **else**
11: **if** $EXT_{ps}(PID \parallel PN_m) < INT_{ps}(PID \parallel PN_n)$ **then**
12: store $EXT_{ps}(PID \parallel PN_m)$ as hidden process into HP.txt
13: m=m+1; $HPC = HPC + 1$; goto step 4
14: **else**
15: **if** $EXT_{ps}(PID \parallel PN_m) > INT_{ps}(PID \parallel PN_n)$ **then**
16: store $INT_{ps}(PID \parallel PN_n)$ as dead process into DP.txt
17: $DPC = DPC + 1$; n=n+1; goto step 4
18: **end if**
19: **end if**
20: **end if**
21: **end if**
22: **end for**
23: **if** $m < p$ &&$n > q$ **then**
24: Store $EXT_{ps}(PID \parallel PN_m)$,...,$EXT_{ps}(PID \parallel PN_p)$ as hidden processes into HP.txt
25: $HPC = HPC + (p\text{-}m)$.
26: **end if**
27: **if** $m > p$ &&$n < q$ **then**
28: Store $INT_{ps}(PID \parallel PN_n)$,..., $INT_{ps}(PID \parallel PN_q)$ as dead processes into DP.txt.
29: DPC=$DPC + (q\text{-}n)$
30: **end if**

The total number of INT_{ps} processes is represented as INT_{psc}

$$INT_{psc} = \sum_{j=1}^{n} PID \parallel PN_j \tag{2}$$

Any inconsistency between EXT_{psc} and INT_{psc} i.e. $EXT_{psc} \neq INT_{psc}$ indicates an abnormal state of the M_{ed-VM}. Algorithm 1 depicts the procedure followed by the $ICVA_P$ to perform the cross-examination between the EXT_{ps} and INT_{ps}. At the end of the scrutiny, $ICVA_P$ provides Hidden Process Count (HPC) and Dead Process Count (DPC), hidden and dead processes.

$$ICVA_P(EXT_{ps}, INT_{ps}) \rightarrow HPC, DPC, hidden, deadprocess \tag{3}$$

To ascertain the hidden and dead processes, the $ICVA_P$ compares the $EXT_{ps}(PID \parallel PN_m)$ with $INT_{ps}(PID \parallel PN_m)$, where $(PID \parallel PN_m)$ is the m^{th} PID and PN. It treats the examined processes as dubious when they are equal. If they are unequal, it checks further to determine whether $EXT_{ps}(PID \parallel PN_m)$ is greater than $INT_{ps}(PID \parallel PN_m)$; if the condition is satisfied, then the $ICVA_P$ declares the $INT_{ps}(PID \parallel PN_m)$ as a dead process. It continues the comparison operation $EXT_{ps}(PID \parallel PN_m)$ with $INT_{ps}(PID \parallel PN_j)$, where j = m + 1, m + 2,..., until it finds that $EXT_{ps}(PID \parallel PN_m)$ is equal to $INT_{ps}(PID \parallel PN_j)$, and then declares the processes from $INT_{ps} = \{PID \parallel PN_m, ..., PID \parallel PN_{j-1}\}$ as dead processes when the condition $EXT_{ps}(PID \parallel PN_m) == INT_{ps}(PID \parallel PN_j)$ is satisfied. If $EXT_{ps}(PID \parallel PN_m)$ is less than $INT_{ps}(PID \parallel PN_m)$, then the $ICVA_p$ declares the $EXT_{ps}(PID \parallel PN_m)$ as a hidden process. The comparison operation between the externally and internally captured state information is continued until all of the elements are examined.

Case 1: $HPC > 0$ indicates that some processes are hidden at the M_{ed-VM}. Equation (4) is an indication of malware infection.

$$((EXT_{psc} \neq INT_{psc}) \&\& (HPC > 0)) \tag{4}$$

Case 2: HPC = 0 denotes that the processes viewed externally are the same as the processes viewed internally. The state of the M_{ed-VM} is dubious state when Eq. 5 is satisfied.

$$((EXT_{psc} == Int_{psc}) \&\& (HPC == 0)) \tag{5}$$

Case 3: The dead process count indicates that the number of processes captured externally is smaller than the number of processes captured internally. This is due to the dynamic nature of the create and destroy of processes. To overcome this situation, A-IntExt first captures the state information of the M_{ed-VM} internally, followed by externally within the TIT.

$$(EXT_{psc} < INT_{psc}) \tag{6}$$

3.4 Online Malware Scanner

The OMS is another key component of the A-IntExt and it performs two key functions. First, from the hypervisor it extracts the complete binary of the hidden process (executable file) that is reported by the $ICVA_P$. The OMS accomplishes

this by utilizing *procdump* plugin of an open source tool[3] on the acquired memory dump of M_{ed-VMs}. For each executable file, it computes three distinct hash digests, such as Message Digest (MD5), Secure Hash Algorithm-1 (SHA-1), and Secure Hash Algorithm-256 (SHA-256). Further, these computed hash digests were checked with Local Malware Database (LMD[4]) to identify any types of hash digests were matched with stored hash digests of known malware types, if not it sends the computed hash digests to powerful public free OMSs and gets an examination report to ascertain whether the extracted executable file is benign or malignant. Similarly, OMS also extracts other processes executable files that are not classified as hidden processes by the $ICVA_P$ that are currently being running in the M_{ed-VM}. These processes are named as dubious processes. Like shrouded processes, the OMS additionally registers hash digest for non-concealed processes and sends them to OMS to identify whether the non-concealed process executable file is malevolent or benign. The procedure involved in determining whether the detected hidden and running dubious (not-hidden) processes are benign or malicious is shown in Fig. 4. The accurate identification and detection of hidden processes leads to A-IntExt generating an alert.

4 Windows VM Memory State Reconstruction

The Intel VT-X and AMD-V virtualization architectures provide hardware-assisted Extended Page Table (EPT) and nested page table to facilitate the address translation more efficiently by leveraging the EPT mechanism [26], the A-IntExt is able to read the guest virtual address from the raw memory contents of the GVM. However, due to the dynamic nature and consistent upgrading of the kernel version, reconstructing the semantic view of the virtual machine is a challenging task for the VMI technology. To reconstruct the memory state of the M_{ed-VM}, the A-IntExt prototype leverages an open-source VMI tool that uses the *xc_map_foreign_range()* function provided in the Xen Control Library (libxc) to understand and reconstruct volatile memory artifacts of the M_{ed-VM} without the consent of the M_{ed-VM}. Later, the same function accesses the RAM memory artifacts, and finally, converts the page frame number to the memory frame number.

Translation of the virtual machine memory address into the corresponding physical address in the host machine is needed to reconstruct the semantic view of the M_{ed-VM}. To reconstruct the memory state information for a commodity operating system (e.g., Windows), the VMI techniques require an in-depth knowledge of the GVM kernel data structures. Static data entries of the kernel symbol table are crucial for the kernel and boot-up procedures. Memory-state reconstruction is the initial step in extracting meaningful high-level information (*ps*, *lsmod*, etc.) from low-level artifacts of the live GVM. This is achieved in

[3] http://www.volatilityfoundation.org/.

[4] LMD consists of 107520 MD5,SHA-1, and SHA-256 hash digest for all previously identified well-known families of malware which was obtained by using https://virusshare.com/ malware repository.

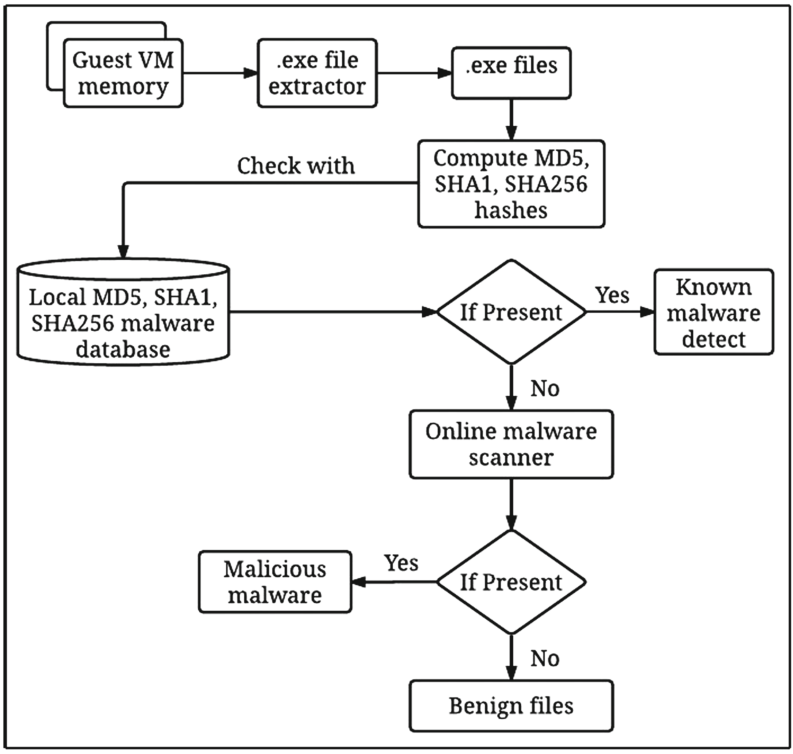


Fig. 4. Online malware scanner

the A-IntExt system by leveraging the VMI technology, while functioning at M_{ing-VM} of the hypervisor with the kernel symbol table of the corresponding GVM.

In the Windows system, each process associated with a data structure is called an *EPROCESS*. Each *EPROCESS* has many data fields and one Forward Link *(FLINK)* pointer and one Backward Link (*BLINK*) pointer. The *FLINK* contains the address of the next *EPROCESS* and *BLINK* stores the address of the previous *EPROCESS*. The first field of the *EPROCESS* is a process control block, which is a structure of type Kernel Process (*KPROCESS*). The *KPROCESS* is used to provide data related to scheduling and time accounting. Other data fields of the *EPROCESS* are PID, Parent PID (PPID), exit status, etc. [24]. The field position of the PID and the PPID in the *EPROCESS* structure may differ from one operating system version to another version, and the series of *FLINK* and *BLINK* systematizes the *EPROCESS* data structures in a circular doubly linked list. A Windows symbol, such as the *PsActiveProcessHead*, points to the doubly linked list. Traversing the *EPROCESS* doubly linked list from the beginning to the end provides all of the running process details.

5 Experimental Results and Evaluation

5.1 Experimental Setup

Experiments were conducted on the host system, which possessed the following specifications: Intel(R) core(TM) i7-3770 CPU@3.40 GHz, 8 GB RAM, and Ubuntu 14.04 (Trusty Tahr) 64-bit operating system. The popular open-source Xen 4.4 bare metal hypervisor was utilized to establish a virtualized environment. To introspect the run state of the live M_{ed-VM}, Windows XP-SP3 32 bit GVM created as DOMU-1 under the Xen hypervisor. The GVM was managed by the trusted VM (DOM-0 i.e. management unit) of the Xen hypervisor. The A-IntExt was installed on the DOM-0 VM, and it leveraged the popular VMI tool, namely, the LibVMI version 0.10.1 to introspect low-level artifacts of the GVMs. The LibVMI traps the hardware events and accesses the vCPU registers, while functioning at the hypervisor.

5.2 Implementation

The implementation of A-IntExt is at three levels: (i) it acts as a VMI system by leveraging a prominent VMI tool to introspect and acquire the GVM running state information without human intervention, (ii) the $ICVA_P$ algorithm is implemented as Proof of Concept (PoC) and induced into the A-IntExt, wherein the $ICVA_P$ detects hidden, dead and dubious processes. In addition, a program was developed that establishes a communication channel between the A-IntExt and the M_{ed-VM}, which also facilitates the transfer of state information by the M_{ed-VM} to the IS_R. (iii) The A-IntExt comprises another major component named OMS (see Sect. 3.4). It is used to identify whether the detected hidden and classified DPs are benign or malicious by auxiliary verification with LMD and large online free public malware scanners[5] while addressing the malicious processes (not hidden) detection challenges as discussed in Sect. 1.

5.3 Windows Malware and Windows Rootkits

To convert the benign Windows GVM into a malicious one and to perform malicious activities on the GVM, two stages of experiments were performed using a combination of both malware and publically available Windows rootkits. In the first stage, the evasive malware variant called Kelihos was directly collected from malware repository[6] to generate bulk malicious processes. In the second stage of the experiment, five publicly available real-world Windows rootkits that have the ability to hide the processes were used.

Experiment 1: Kelihos is a Windows malware also known as Hlux. Once it starts to execute, it generates a number of child processes, and then exits from the main process to conceal its existence. It launches a set of processes in a span

[5] https://www.virustotal.com/.
[6] http://openmalware.org/.

of a short interval, which influences the process count. The main function of the generated child process is to monitor user activities, and then report it to the Command and Control Server (C&C) to be joined into a botnet. The Kelihos malware was used to breed a number of processes, and at the same time, the Hacker defender rootkit was used to hide the process. This test was done to demonstrate the detection accuracy of the A-IntExt under a dynamic process creation environment. The A-IntExt extracts the manipulated semantic kernel data structure details related to the process by walking through the *EPROCESS* data structure and its associated *PsActiveProcessHead* symbol (see Sect. 4).

Table 2. Detection and classification of hidden, dead and DPs by the A-IntExt for windows GVM

Exp	PS used	PS visible at GVM	PS Introspected by A-IntExt	No. of PS classified by A-IntExt			Time in (Sec)
				HPC	DPC	DPs	
Test-1	25	20	25	5	0	20	0.22
Test-2	50	45	50	5	0	45	0.41
Test-3	75	70	74	5	1	69	0.63
Test-4	100	95	99	5	1	94	0.82
Test-5	125	120	123	5	2	118	1.03

The $ICVA_p$ is a subcomponent of the A-IntExt and its task is to identify hidden, dead and dubious processes by performing a comparison operation on the internally and externally captured state information of the M_{ed-VM}. The performance evaluation tests for both the $ICVA_p$ and the A-IntExt were conducted separately. To measure the execution speed of the $ICVA_P$ in detecting the hidden and dead processes, experiments were performed with different numbers of processes, i.e., 25, 50, 75, 100, and 125. The execution speed denotes the amount of time the $ICVA_P$ takes to derive a conclusion as to whether the process is hidden, dead or dubious processes. The last columns of Table 2 depicts the average detection time of the $ICVA_P$ for different numbers of processes on the Windows GVM. One can observe that the detection time of the $ICVA_P$ for 125 processes is less than 1.03 s.

Table 3. Identifying an actual malicious process from detected hidden processes by OMS of A-IntExt on Windows GVM.

Exp	No. of HP	Computed MD5 hash for classified HP	Checked as	PS name	D R
1	5	55cc1769cef44910bd91b7b73dee1f6c	Malicious	hxdef073.exe	37/53
		be046bab4a23f8db568535aaea565f87	NF	procdump.exe	0/53
		6cf0acd321c93eb978c4908deb79b7fb	NF	chrome.exe	0/53
		bf4177e1ee0290c97dbc796e37d9dc75	NF	iexplore.exe	0/53
		d068da81e1ab27dc330af91bffd36e6b	NF	firefox.exe	0/53

Table 4. Identifying an actual malicious processes from detected and classified DPs by OMS of A-IntExt on Windows GVM.

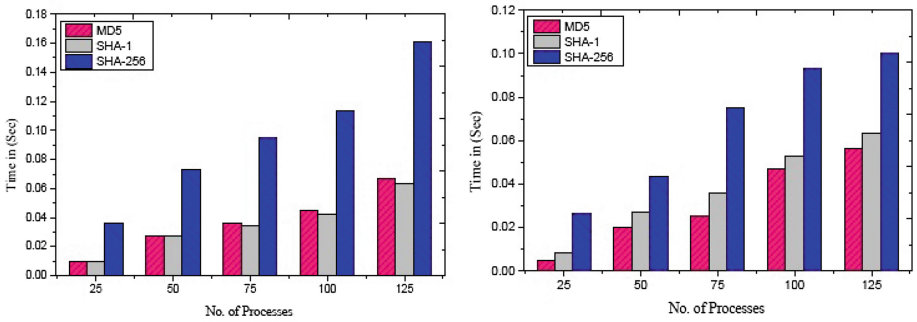
Exp	No.of DPs	Scanned result		Malicous PS reported with MD5 hash	Name	D R
		Benign	Malware			
1	20	18	2	0bf067750c7406cf3373525dd09c293c	EFMTnkT7m.exe	–
				5fcfe2ca8f6b8d93bda9b7933763002a	kelihos dec.exe	37/55

Twenty-five processes were considered in the first test; each test was performed five times to derive the average detection time. Prior to the evaluation, five processes were hidden at the M_{ed-VM} and all of them were correctly detected by the A-IntExt, including the hidden, dead, and DPs, as shown in Table 2. Further, A-IntExt precisely address the malicious process detection challenges (see Sect. 1) by leveraging its OMS component. As part of the experimental observations, Test-1 of Table 2 describes the 25 processes externally introspected by A-IntExt, which includes five hidden processes and twenty DPs that are classified by the $ICVA_P$; these hidden and DPs are propagated by the malware. In our experiment-1, we used kelihos malware to generate malicious processes (not hidden) and perform spiteful activity on M_{ed-VM}. At the same time, we used hacker defender rootkit to hide some processes. During introspection of the untrustworthy M_{ed-VM}, A-IntExt precisely classified the infection activity of the malware processes as hidden and DPs. Table 3 describes that from the five detected hidden processes, one process (*hxdef073.exe*) is correctly identified as malicious with Detection Rate(DR) of 37/53 based on the computed hash, and the other four processes such as the *procdump.exe*, *chrome.exe*, *iexplorer.exe*, and *firefox.exe*, which were actually hidden by the hacker defender rootkit, are reported as benign by the OMS. Similarly, Table 4 represents the 20 DPs that were classified by A-IntExt further, those processes were checked with both LMD and OMS based on the computed hashes. The time taken to compute MD5,SHA-1,SHA-256 hashes and cross-check with LMD are depicted in Fig. 5. As a result, one process ($EFMTnkT7m.exe$) is identified as malicious by locally checking with LMD (without forwarding to virustotal) and other advanced malware process ($kelihos\_dec.exe$) identified as malicous checking with OMS as shown in Fig. 6, and the rest were recognized as benign or Nothing Found (NF).

Table 5. List and functionality of Windows rootkit

Rootkit name	User mode/Kernel mode	Target object	Hide PS	Detected by A-IntExt
Fu Rootkit	Kernel mode	EPPROCESS	Yes	Yes
HE4Hook	Kernel Mode	EPPROCESS	Yes	Yes
Vanquish(0.2.1)	User mode	IAT,DLL	Yes	Yes
Hacker Defender	User mode	IAT,DLL	Yes	Yes
AFX Rootkit	User mode	IAT,DLL	Yes	Yes

IAT: Interrupt Address Table, DLL: Dynamic Link Library



(a) (b)

Fig. 5. The average time taken by OMS to compute MD5,SHA-1, and SHA-256 hashes for different processes (5a). Time taken by OMS to detect malware by cross-checking with LMD based on it's computed hashes (5b).

Experiment 2: In the second stage of the experiment, five publicly available Windows rootkits were used as shown in Table 5. The third and fourth columns of Table 5 represent target object and complete functionality of the rootkit, respectively. However, in this stage of the experiment, the detection capability of the A-IntExt was limited to only the processes. For example, the *FU rootkit* leverages the direct kernel object manipulation technique to hide a list of active processes by directly unlinking the doubly linked list *EPROCESS* data structure. It contains the *fu.exe* executable file and the *msdirectx.sys* system file. The function of hiding the kernel driver module files is achieved by the *msdirectx.sys*, whereas the *fu.exe* file is used to configure and command the driver. The *FU rootkit* is capable of achieving privilege escalation of the running processes and can also alter the DLL semantic object of the kernel data structure by rewriting the kernel memory. The*HE4Hook* is a kernel-mode rootkit and the user-mode rootkits are*Vanquish*, *Hacker defender*, and *AFX Rootkit*. These rootkits have the potential to hide the running processes on the Windows system. The fifth column of Table 5 represents the detection of hidden processes performed by the A-IntExt.

6 Performance Overhead

A series of tests were conducted using Windows system benchmark tools to determine the performance impact of the A-IntExt. The benchmark tests were executed on the Windows GVM in two different scenarios to evaluate the performance impact of the A-IntExt. In the first scenario, the A-IntExt was disabled (not functioning), and in the second scenario the A-IntExt was enabled (running). PCMark05, an industry standard benchmark, was executed on the Windows GVM to quantify the performance impact of the A-IntExt. Tests such as the CPU, Memory, and HDD of the PCMark05 suite were considered. These

```
OMS scan results for MD5 hash:
-------------------------------------------------------------------------------------------------------
MD5 value: 5fcfe2ca8f6b8d93bda9b7933763002a
Online Malware Scanner (OMS)  scan date: 2016-07-02 06:34:51
VirusTotal engine detections: 37/55
Link to VirusTotal report:
https://www.virustotal.com/en/file/9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911
d22/analysis/
-------------------------------------------------------------------------------------------------------
OMS scan results for SHA-1 hash:
-------------------------------------------------------------------------------------------------------
SHA-1 value: 581c0425386c44b6056b66dbe36d50aefd4ca724
Online Malware Scanner (OMS)  scan date: 2016-07-02 06:34:51
VirusTotal engine detections: 37/55
Link to VirusTotal report:
https://www.virustotal.com/en/file/9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911
d22/analysis/
-------------------------------------------------------------------------------------------------------
OMS scan results for SHA-256 hash:
-------------------------------------------------------------------------------------------------------
SHA-256 value: 9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911d22
Online Malware Scanner (OMS)  scan date: 2016-07-02 06:34:51
VirusTotal engine detections: 37/55
Link to VirusTotal report:
https://www.virustotal.com/en/file/9d48503fa42f4184873fbc796a2fb64ad61eb9fcb7a1647a4830aceaf9911
d22/analysis/
-------------------------------------------------------------------------------------------------------
```

Fig. 6. OMS results for *kelihos_dec.exe* malware

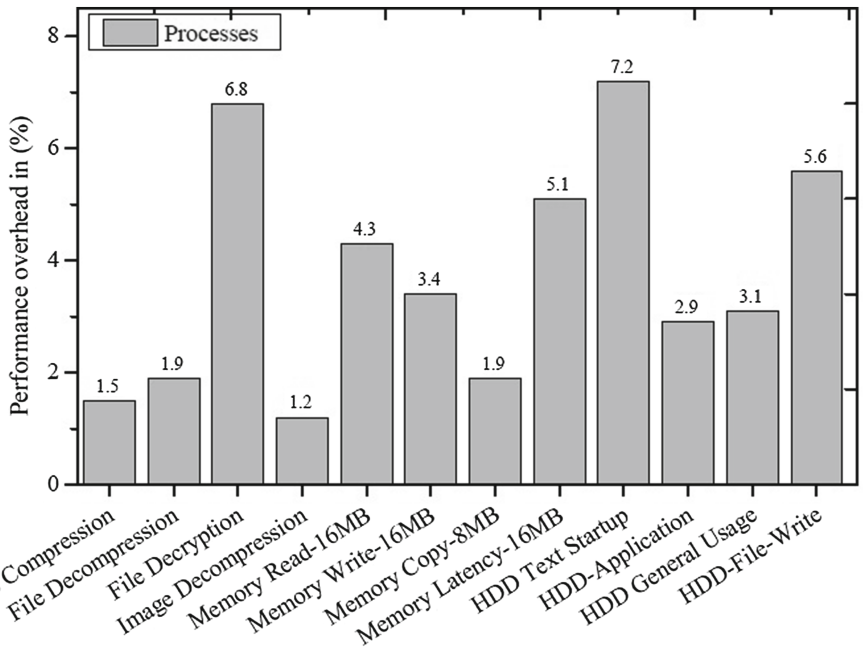


Fig. 7. Performance impact of A-IntExt on PCMark05 in detecting hidden and malicious state information of M_{ed-VM} for Windows GVM

tests were executed separately five time on the GVM. Finally, the results were considered on an average five-time execution of each test.

During hidden, dead and DPs process detection, tests such as File Decryption, HDD-Text Startup, and HDD-File-Write induced maximum performance overheads of 6.8 %, 7.2 %, and 5.6 %, respectively, other tests performance overheads observed is less than 5.5 %. These were noticed while the A-IntExt performed process introspection traces on the executed malware and rootkits. Figure 7 represents the overall performance of the A-IntExt in detecting hidden, dead and dubious processes detection.

The main reason for the performance loss is due to direct introspection and the semantic view reconstruction operation performed by the A-IntExt. As the $ICVA_P$ achieve the job offline, there is no performance overhead.

7 Discussion

The existing VMI techniques facilitate reconstructing a few semantic views of the M_{ed-VM} by directly intercepting the RAM contents of the live M_{ed-VM} by overcoming the semantic gap problem. However, these techniques are yet to be intelligent and automated to introspect and accurately detect hidden or malicious semantic state information on their reconstructed high-level semantic view. The design, implementation, and evolution of the proposed A-IntExt are signified as an intelligent solution to precisely detect the malignant processes running on the M_{ed-VM}. It acts as a perfect VMI-based malware symptoms detector by logically analyzing the malicious infection of the operating systems key source information (processes). The $ICVA_p$ of the A-IntExt judiciously performs a cross-examination to detect the hidden-state information of the GOS that is manipulated by different types of evasive malware or stealthy rootkits. Malicious processes (not-hidden) are identified by the OMS. We believe that the current development of A-IntExt is proficient in detecting hidden, dead, and malicious processes of any kind of malware or rootkit. However, detecting and identifying both known and unknown malware processes by performing cross-examination with both LMD and powerful online malicious content scanners (Viroustotal) using it's computed hashes (MD5, SHA-1, and SHA-256). The major limitation in identifying malicious processes by the Viroustotal is that it accepts only four requests per minute.

8 Conclusion and Future Work

In this work, we designed, implemented, and evaluated the A-IntExt system, which detects hidden, dead and malicious processes by performing an intelligent cross-view analysis on the internally and externally captured run-state information of the M_{ed-VM}. The A-IntExt abstracts the semantic view (processes) of the live Windows GVM externally (VMM-level). It uses an established communication channel between the M_{ing-VM} and M_{ed-VM} to receive internally captured run-state information (at-VM-level), further proficiently detecting hidden and malignant state information of the M_{ed-VM} that could be manipulated by sophisticated malware or real-world rootkits. The A-IntExt is intelligent enough

to address the challenges that lie in detecting malicious (not-hidden) processes of the run state of the M_{ed-VM} using its OMS component. Publicly available evasive malware, real-world Windows rootkits were used to perform a series of experiments to accurately measure the hidden-state and malicious detection capability of the A-IntExt. The experimental results showed the accuracy of A-IntExt in detecting stealthy processes with a maximum performance overhead of 7.2 %.

As future work, we plan to enhance the detection capability of the A-IntExt to detect unknown malware which are not recognized by OMS of A-IntExt, by incorporating machine learning algorithms so that detection capability A-IntExt can be evaluated against most commonly emerging advanced persistent threats, elusive malware and rootkit.

References

1. Pearce, M., Zeadally, S., Hunt, R.: Virtualization: Issues, security threats, and solutions. ACM Comput. Surv. (CSUR) **45**(2), 17 (2013)
2. Barford, P., Yegneswaran, V.: An inside look at botnets. Malware Detection. Springer, New York (2007)
3. Lanzi, A., Sharif, M.I., Lee, W.: K-Tracer: a system for extracting kernel malware behavior. In: NDSS (2009)
4. Prakash, A., et al.: Manipulating semantic values in kernel data structures: attack assessments and implications. In: 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). IEEE (2013)
5. Jiang, X., Wang, X., Dongyan, X.: Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM (2007)
6. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: NDSS. vol. 3 (2003)
7. Payne, B.D., Martim, D.D.A., Lee, W.: Secure and flexible monitoring of virtual machines. In: Twenty-Third Annual Computer Security Applications Conference, ACSAC 2007. IEEE (2007)
8. Srinivasan, D., et al.: Process out-grafting: an efficient out-of-VM approach for fine-grained process execution monitoring. In: Proceedings of the 18th ACM Conference on Computer and Communications Security. ACM (2011)
9. Dolan-Gavitt, B., et al.: Virtuoso: narrowing the semantic gap in virtual machine introspection. In: 2011 IEEE Symposium on Security and Privacy. IEEE (2011)
10. Jain, B., et al.: SoK: introspections on trust and the semantic gap. In: 2014 IEEE Symposium on Security and Privacy. IEEE (2014)
11. Fu, Y., Lin, Z.: Bridging the semantic gap in virtual machine introspection via online kernel data redirection. ACM Trans. Inf. Syst. Secur. (TISSEC) **16**(2), 7 (2013)
12. Saberi, A., Yangchun, F., Lin, Z.: HYBRID-BRIDGE: Efficiently bridging the semantic gap in virtual machine introspection via decoupled execution and training memoization. In: Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS-2014) (2014)
13. Bauman, E., Ayoade, G., Lin, Z.: A Survey on Hypervisor-Based Monitoring: approaches, applications, and evolutions. ACM Comput. Surv. (CSUR) **48**(1), 10 (2015)

14. Goudey, H.: Threat Report: Rootkits. https://www.microsoft.com/en-in/download/details.aspx?id=34797
15. Xuan, C., Copeland, J., Beyah, R.: Toward revealing kernel malware behavior in virtual execution environments. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 304–325. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04342-0_16
16. Jones, S.T., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: VMM-based hidden process detection and identification using Lycosid. In: Proceedings of the fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. ACM (2008)
17. Richer, T.J., Neale, G., Osborne, G.: On the effectiveness of virtualisation assisted view comparison for rootkit detection. In: Proceedings of the 13th Australasian Information Security Conference (AISC 2015), vol. 27, p. 30 (2015)
18. Wu, R., et al.: System call redirection: A practical approach to meeting real-world virtual machine introspection needs. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE (2014)
19. Westphal, F., et al.: VMI-PL: a monitoring language for virtual platforms using virtual machine introspection. Digital Invest. **11**, S85–S94 (2014)
20. Fu, Y., Zeng, J., Lin, Z.: HYPERSHELL: a practical hypervisor layer guest OS shell for automated in-VM management. In: 2014 USENIX Annual Technical Conference (USENIX ATC 2014) (2014)
21. Jones, S.T., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Antfarm: tracking processes in a virtual machine environment. In: USENIX Annual Technical Conference, General Track (2006)
22. Litty, L., Andres Lagar-Cavilla, H., Lie, D.: Hypervisor support for identifying covertly executing binaries. In: USENIX Security Symposium (2008)
23. Wang, Y.-M., et al.: Detecting stealth software with strider ghostbuster. 2005 International Conference on Dependable Systems and Networks (DSN 2005). IEEE (2005)
24. Lamps, J., Palmer, I., Sprabery, R.: WinWizard: expanding Xen with a LibVMI intrusion detection tool. In: 2014 IEEE 7th International Conference on Cloud Computing. IEEE (2014)
25. Vmware, 2011. Vmware, inc. vprobes programming reference. http://www.vmware.com/pdf/ws8_f4_vprobes_reference.pdf
26. Aneja, A.: Xen hypervisor case study-designing embedded virtualized Intel architecture platforms. Intel, March 2011. https://www.intel.in/content/dam/www/public/us/en/documents/white-papers/ia-embedded-virtualized-hypervisor-paper.pdf