Virtual Machine Introspection based Spurious Process Detection in Virtualized Cloud Computing Environment

Ajay Kumara M.A.
Department of Information Technology
National Institute of Technology, Karnataka
Surathkal, Mangalore, India
ajayit13f01@nitk.edu.in

Jaidhar C.D
Department of Information Technology
National Institute of Technology, Karnataka
Surathkal, Mangalore, India
jaidharcd@nitk.ac.in

***Abstract*—Virtual Machines are prime target for adversary to take control by exploiting the identified vulnerability present in it. Due to increasing number of Advanced Persistent Attacks such as malware, rootkit, spyware etc., virtual machine protection is highly challenging task. The key element of Advanced Persistent Threat is rootkit that provides stealthy control of underlining Operating System (kernel). Protecting individual guest operating system by using antivirus and commercial security defense mechanism is cost effective and ineffective for virtualized environment. To solve this problem, Virtual Machine Introspection has emerged as one of the promising approaches to secure the state of the virtual machine. Virtual Machine Introspection inspects the state of multiple virtual machines by operating outside the virtual machine i.e. at hypervisor level. In this work, Virtual Machine Introspection based malicious process detection approach is proposed. It extracts the high level information such as system call details, opened known backdoor ports from introspected memory to identify the spurious process. It triggers an alert in response to detected intrusion.**

Keywords—Hypervisor; Semantic Gap; Rootkit; Virtualization; Virtual Machine; Virtual Machine Introspection;

I. INTRODUCTION

Virtual Machine (VM) in the context of virtualized cloud infrastructure poses greater security challenges as adversary cloud easily rent VM from Cloud Service Provider (CSP). Hypervisor is a pillar of cloud computing [1], it abstracts physical hardware resources to multiple Guest Operating System (GOS) which are running on the hypervisor. If any exploitable vulnerabilities present in hypervisor, attacker could compromises entire virtualized cloud infrastructure by considering virtual machine as primary target. Securing different types and version of guest operating system (VM) from advanced persistent threats such as malware, rootkit and spyware etc., is massive challenges for CSP.

The current generation of anti-malware measure such as antivirus, agent based Intrusion Detection System (IDS) are signature dependent and they are trying to protect themselves instead of beating persistent advanced malware (e.g. Agobot variant rootkit [2]). In addition, signature based detection fails to detect unknown attacks without the corresponding signature in the signature database. Such types of defence mechanisms are ineffective to secure virtual machines in order to overcome

This issue, Virtual Machine Introspection (VMI) is promising approach to monitor the run state of the VM by operating outside the virtual machine i.e. at hypervisor. Virtualization is a powerful platform for abstraction of physical hardware resources to guest operating system by encapsulating software layer hypervisor. Virtual Machine Monitor (VMM) is also called as hypervisor. Rapid emulation of commodity physical hardware also allows the hypervisor to share the resources logically among the virtual machines. It increases the greater security risk and becoming an attractive target for sophisticated attacker.

Virtual Machine Introspection is a fine-grained approach. Isolation property of VMI allows to inspect Memory, Disk, CPU registers, Network connections and other related hardware events [3] of virtual machine during run state from hypervisor level. Securing virtual machine using VMI is an active research area. Researchers have proposed number of solutions by realizing efficiency of VMI technique for real world persistent intrusion detection [3] [4], malware analysis [5], memory forensics [6] etc.

Numbers of open source VMI tools are developed to introspect the run state of virtual machine. One such prototype is Liveware [3] that is able to view virtual machine state during running time. Open source tool called XenAccess [7] inherits the property of VMI and it inspects memory and CPU registers of virtual machines running on Xen Hypervisor. Extended version of XenAccess named as LibVMI supports both well-known open source hypervisors namely Xen [8] and KVM [9]. Similarly, virtual machine disk inspection achieved by LIbGuestFS [10]. Similar approach adopted by VMware EXSI hypervisor is VMsafe [11] that introspects CPU Registers, Memory and Disk contents of VMware based virtual machine. Many researchers actively leveraging the VMI to provide robust security and identify abnormal activities of virtual machine [4] [5] [6].

The main limitation of VMI called as semantic gap: extraction of high-level information of guest operating system

978-1-4799-8433-6/15/$31.00©2015 IEEE

state from low-level artifacts obtained externally at hypervisor level (e.g. Active process, loadable kernel module, guest system calls, hyper-calls and network connections etc.). Meaningful information extraction from raw data of the virtual machine memory is challenging as it needs in-detailed knowledge of the guest operating system [12] [13] [14]. Generally VMI uses guest operating system symbol table (Sytem.map in Linux and ntdll.dll in windows) to interpret the low level details of guest virtual machine. GOS symbol table provides the information related to system calls and Interrupt Descriptor Table (IDT) [17]. If any changes to guest operating system code or application code either by malware or rootkit or attack causes daunting task to hypervisor [14]. Researchers have proposed variety of solutions for malware detection and rootkit detection for cloud computing environment [5]. Only few work have been proposed based on system call approach [13] [26].

In this paper, Virtual Machine Introspection based malicious process detection approach is proposed. Detection is based on extracted system call information. In addition, opened known backdoor ports also used to identity spiteful processes.

The remainder of this paper is organized as follows: section II provides the background for hypervisor and VMI. Section III gives the related work. Section IV represents overview of our proposed work. Outline of LibVMI tool described in Section V. Section VI provides experimental setup and preliminary results. Section VII considers scope and assumption. Finally, conclusion and future work illustrated in section VIII.

II. BACKGROUND

A. Hypervisor

Hypervisors is also referred to as Virtual Machine Monitor (VMM). It is a software program that emulates or abstract physical hardware resources such as (CPU, Memory, Disk, Network Interface Card, I/O) to a multiple guest operating system which are running on the same physical machine. Hypervisor has full control over entire virtual resources of the virtual machines. In general, hypervisor act as bedrock between virtual machine and physical hardware. Hypervisor broadly classified into two types: Type 1 hypervisor and Type 2 hypervisor. The Type1 hypervisors run directly on the system hardware. Type2 hypervisors run on a Host Operating System (HOS) and provides virtualization services, namely I/O device support, memory management etc., to multiple virtual machines [15].

B. Xen Hypervisor

Xen [8] is popular Type1 bare metal hypervisor and its current version is xen 4.5.0. It uses the most privileged domain Dom0 or Domain0 as a management unit to map or manage virtualized resources to virtual machines which are emulated by the Xen hypervisor from bare hardware. Xen supports full virtualization as well as para virtualization. Hardware Assisted Virtualization (HAV) also referred as para virtualization. In case of full virtualization, virtual machine completely abstracted from the underlying hardware. GOS utterly unaware its virtualized state. In HAV, the technology "Intel VT-x" and "AMD-V" allows to run without modifying OS. Para-virtualization allows the software interface (API) that lies between virtual machine and hypervisor to modify the requair-

ed operating system kernel and also replace non-virtualizable instructions into hyper-calls. It aware of its virtualized state.

C. Virtual Machine Introspection

Traditional host based and network based intrusion detection tools are ineffective in detecting and eradicating sophisticated attacks in a timely manner. Since their detection capability is restricted to single devices, they are inefficient to protect the entire cloud computing infrastructure against new persistent threats. Fig. 1 shows an overview of agent based malware/intrusion detection method, it does not provide an accurate view of the internal behavior of guest operating system. Some of the defense tools which are functioning in the virtual machine are targeted and compromised by advanced malware (e.g., Agobot variant rootkit compromise 101 anti-malware tools [2]). This limitation is overcome by Virtual Machine Introspection. Indirect inspection of target virtual machine run state is stealthily achieved through a VMI approach by functioning at hypervisor level in a virtualized environment [16]. Fig. 2 depicts a generic view of Virtual Machine Introspection. It uses hypervisor specific Application Programming Interface (API) and guest operating system kernel symbol table (*system.map in Linux, ntdll.dll in windows*) [17] to make sense low-level state information of virtual machine. It obtains symbol table including IDT and the crucial kernel data structure of the guest operating system.

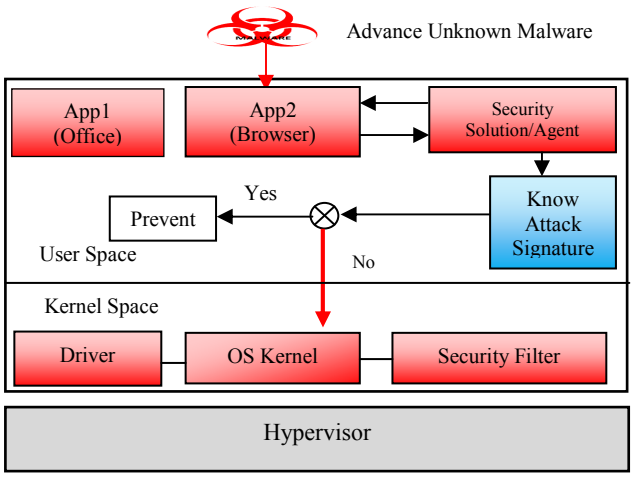

Fig. 1 Host or Agent based Security Solution

D. VMI based Virtual Machine Memory Inspection

In order to provide high performance, greater availability and assure robust security for multiple guest operating systems, protecting and monitoring of virtual machine memory is highly crucial. VMI based memory forensic is one such technique to perform this task at hypervisor level [18]. Python based volatility forensic memory analysis framework

support for different guest operating system (like Mac OS, Linux and Windows). It extracts the operating system data structure from memory pages of introspected virtual machine. Identification of legitimate running process and hidden process details could help to spot malicious process created by rootkit in a VM. Memory multiplexing degrade the performance of virtual machines [19]. VMI is able to access free memory pool information obtained from a different virtual machine during runtime. This increases the speed by reduction of high traffic load associated with memory. Virtual memory ballooning [20] effectively address this issue.

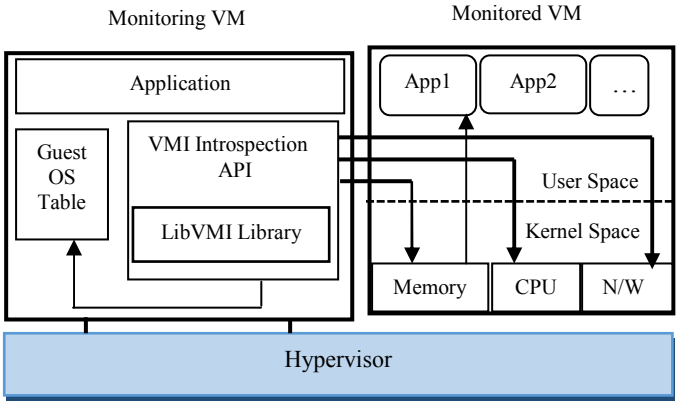

Fig. 2 Generic Architecture of Virtual Machine Introspection

Digital forensic community address the VMI semantic gap problem from the prospective of Forensic Memory Analysis (FMA) [21]. Simple software used in this technique to access the guest operating system memory. Detailed analysis of the internal activities of guest virtual machine memory benefited by forensic. This prototype implemented on Xen hypervisor using XenAccess or (LibVMI library). It withstands the guest virtual machine memory contents against sophisticated memory rootkits.

The Virtual Machine Introspection technique has been widely used to build robust security tool for inspection of malignant activities of virtual machines including intrusion detection system, memory forensic [6]. Maintaining the integrity of guest operating system kernel data structure or memory is a highly challenging task due to multiple application running on virtual machine. For instance attacker cloud run malicious applications to exploit the vulnerabilities of VM to compromise the VM.

III. Related Work

The Virtual Machine Introspection technique has played vital role in securing virtual machine with number of security solution. The idea proposed by Garfinkel and Rosenblum [3] detects intrusion at run state of virtual machine by positioning Intrusion Detection System (IDS) externally at hypervisor level. This idea implemented as a prototype Livewire [3]. Later on a number of solutions have proposed in this direction for different aspects including malware detection [5] memory forensics [6] etc. Virtuoso [12] is one such approach to bridge the semantic gap of VMI by automatically creating introspection tool with that can semantically extract meaningful information of guest operating system based on low level data source using dynamic slicing algorithm. The main limitation of Virtuoso is that it is not fully automated and it requires minimal human effort. The same approach extended by Yangchuan Fu and Zhiquiang as VMST [13] and it address the semantic gap of VMI by enabling automatic generation of secure introspection tool with a number of new features and capabilities. It significantly eliminates the limitation of virtuoso involvement from the end-user. Memory forensic tools facilitate with virtual machine introspection. Dolan-Gavitt et. al. [21] adopted the memory forensic technique to inspect the run state of virtual machine memory contents by creating simple software that is able to view memory contents accurately. It cloud immediately understand by forensic tool to allow the VMI to proceed the inspection of VM memory state in deep.

Virtual introspection for XEN (VIX) tool developed by Hay and Nance [6] for Xen hypervisor based virtual machine. It allows an investigator to perform live analysis of un-privileged virtual machine (DomU) memory content from a privileged monitoring virtual machine (Dom0) on Xen hypervisor. VIX consists of Linux based command utility such as (ps, lsmod, netstat, lsof, who, and top). It also address the significant challenges of recovery of volatile memory of the virtual machine. Liu et.al [27] proposed an approach called TxIntro that leverage the Hardware Transactional Memory (HTM) for concurrent, consistent and timely introspection of guest operating system by actively monitoring critical kernel data structure of the monitored system.

Hizver et.al [18] proposed real-time kernel data structure monitoring (RTKDSM) system of VM that simplifies and automate the analysis of execution state of virtual machine. RTKDSM system bridge the semantic gap of VMI by leveraging the vast kernel data structure of guest operating system using existing open source computer forensics framework called volatility.

IV. Proposed Work

Our proposed work that detects malicious process based on system calls and opened backdoor ports information is described in this section. It adopts Virtual Machine Introspection technique to spot the malicious events of virtual machines in virtualized cloud environment by identifying malicious behavior of process. Further, it also verify whether any known backdoor ports are opened or not. Major components of our proposed architecture are state information extraction module, intrusion detection module and notification module. Fig.3 depicts an overview of the proposed architecture.

State Information Extraction Module: is the workhorse of the proposed work. It operates at hypervisor level as an independent entity and maps raw memory pages of monitored virtual machine (domU) into monitoring virtual machine (dom0). It reconstruct the memory from introspected raw data

to deduce high level information such as list of running process, system call information, active network connections and opened files etc. This is achieved by knowing the algorithms as well as data structure adopted by the guest operating system. Monitored virtual machine kernel coding semantic can also be used to derive the meaningful information from raw memory pages. In Linux, the head of the process list is *init_task* is statically declared task. Doubly linked list data structure is used to interlink the *task_struct* (process list). Traversing from the head of the process list provides process identifier "*pid*", "*process name*" and pointer

order. Deviation in system call order witness for abnormal behavior of process. It signals the notification module as and when anomaly is observed.

Malicious Port Detection Engine: holds database of well-known backdoor ports. For instance Blackhole uses TCP port Number 12345 as a backdoor [22]. Gummo runs on TCP port Number 31337[22]. Bdoor runs on TCP port number 8080. [22]. Oas0n runs on TCP port 2936 [22]. Received source port and destination port numbers are compared with backdoor database to ascertain the opened backdoor ports by malicious software. It signals the notification module whenever match is observed.

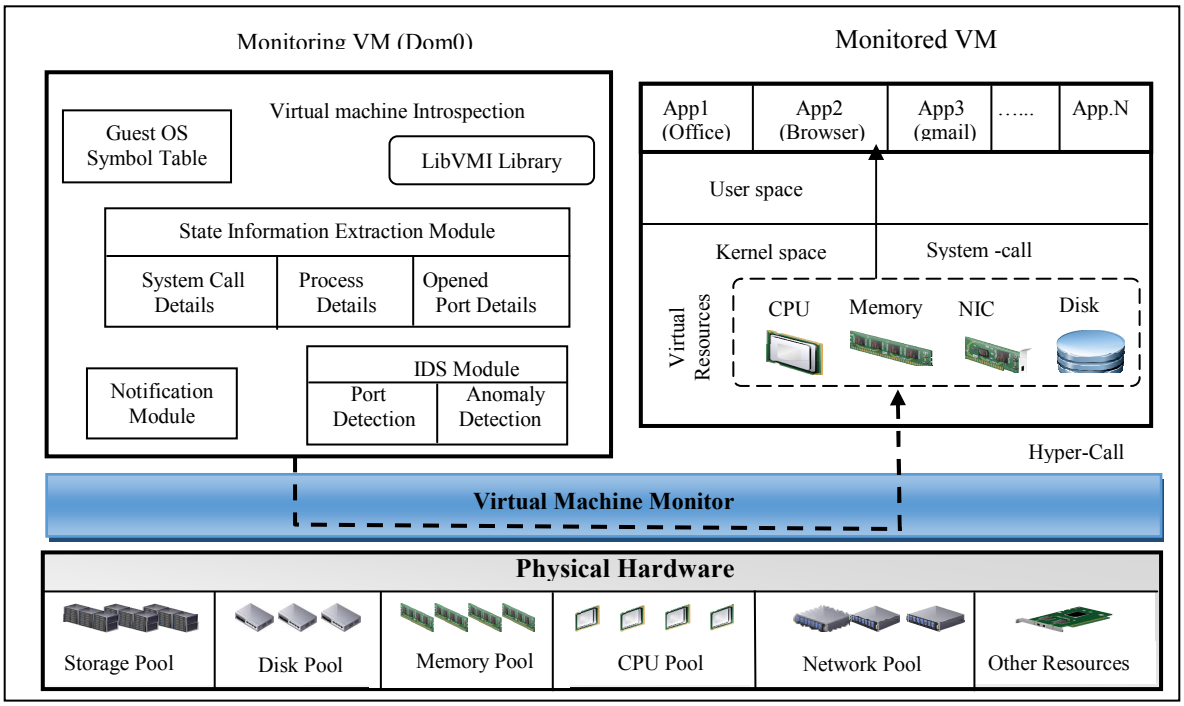

Fig. 3. Proposed Work of Virtual Machine Introspection on Xen Hypervisor

to the next process. Extracted information are transferred to intrusion detection module for analysis.

Intrusion Detection Module: purpose is to collect and analyze the data received from state information extraction module. It analyze the data in real time. Its sub module are: a) Anomaly based detection engine b) malicious port detection engine.

Anomaly based Detection Engine: maintains base profile of each system call including its parameter details. In addition, it maintains a base profile of each process which provides a relation between process and system call (which process is eligible to invoke what system calls including their order). In other words, it has a complete knowledge of which process can raise what system calls including the order of system calls. Introspected system call including parameters are compared with system call base profile database. Comparison discrepancy signifies the abnormal behavior. It checks which process has initiated what system calls and their

Notification Module: function is to generate an alert as an indication that system has detected the potential attack. The notification message can be an email, audio signal, siren, popup message or in a log file.

V.VM MEMORY INTROSPECTION USING LIBVMI TOOL

The primary goal of LibVMI [8] library focused on reading and writing memory from virtual machines (VMs). It is also provides the functionality for accessing CPU registers, network statistics etc., of the virtual machine. LibVMI tool currently compatible for XEN hypervisor as well as KVM hypervisor. In our work LibVMI tool particularly installed on privileged management of XEN hypervisor i.e. (Dom0) for memory access of the virtual machine (DomU). LibVMI consisting of many memory access utilities [23] of targeted virtual machine. For instances, vmi ***read addr ksym:*** function reads the specified kernel symbol value from the

targeted guest operating system that will be converting as virtual address by using system symbol map. The **vmi_read_addr_pa:** function reads count bytes from memory located at a specified physical address in a target guest operating system and stores the output in a buffer. Similarly the **vmi_read_addr_va** read the count bytes from the memory presented at virtual address of the target guest operating system and results stored in a buffer. The write function of **vmi_write_addr_ksym**: write target values to the target operating system. These functions represent the overview of the LibVMI API related to memory introspection of targeted virtual machine [23].The same design of LibVMI extended that includes disk, network information introspection in a similar way.

VI. EXPERIMENTAL SETUP AND PRELIMINARY RESULT

To establish the experimental setup, we have installed Xen (4.1) hypervisor on Ubuntu 12.04 (Precise Pangolin) 64bit operating system. The host system has Intel corei7 processor with 8GB memory. The VMI tool LibVMI version 0.10.1 has installed on the most privileged domain (Dom0) or management unit of the Xen hypervisor. Virtual machine launched by the Xen hypervisor as DomU has Ubuntu 12.04 Guest Operating System as shown in fig 6. LibVMI-0.10.1 has used to view the run state of Ubuntu 12.04 LTS virtual machine. Two popular real world Linux based rootkits such as Average Coder rootkit [24] and Jynx2 rootktik [25] have used in this experiment to convert the benign virtual machine into malicious one.

Fig . 4.1 Jynx2 Rootkit files with User defined folder "abc"

Fig..4.2 Jynx2 Rootkit file Compilation

Fig. 4.3 Jynx2 Rootkit Injected and System Turns Abnormal

Fig. 4.4 "abc" file hidden by Jynx2Rootkit

Fig. 4. Experimental results of Jynx2 Rootkit on Xen based VM

The Process behaves abnormally once they are under the control of the malware (Rootkit). One of the ways to detect process abnormal behavior is through careful analysis of system call such as system call sequence, number of parameters, raised by which process, whether process is authorized to invoke the system call. The result of the analysis helps in detecting malicious process.

A. Preliminary result analysis against real world malware

Jynx2 [25] is Linux based user-mode rootkit. It dynamically inserts malicious library function into system binaries during runtime. It does not replace the original binary files of the system to evade the detection from digest based detection method. Once Jynx2 rootkit program is executed, the system checks for shared libraries that are needed to be loaded at run time by referring /etc/ld.so.conf and /etc/ld.so.preload files. The LD_PRELOAD is an environment variable may be used to point to a shared library (added by Jynx2 rootkit). Using this feature, Jynx2 rootkit creates a malicious shared library to hide the files, processes and ports. A utility like "ps", "netstat" and "ls" loads malicious shared library function into running process. As a result, files, processes, network connections and ports used by malware are hidden. Malignant user or attacker can view the processes, files and network connection details through a shared library created by the rootkit. Jynx2 rootkit successfully injected on Ubuntu 12.04LTS VM. Fig. 4 demonstrates the shared libraries of "ps" utility before and after the injection of a rootkit. The file"/etc/ld.so.preload" is hidden due to the impact of a rootkit.

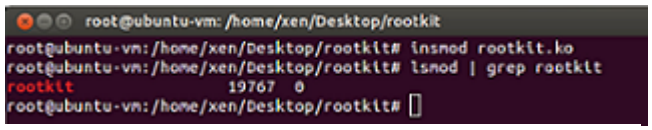

Fig. 5.1 Average Coder Rootkit Injected ontoVM

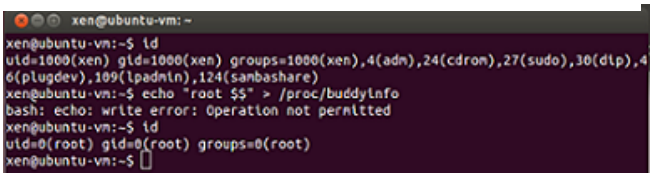

Fig. 5.2 Average Coder Rootkit hiding Process Details

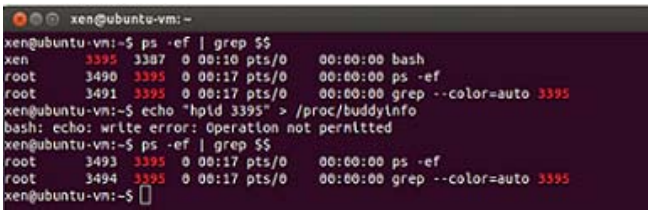

Fig. 5.3 Privileged Permission Restricted by Rookit

Fig. 5. Experimental result of Avarage Coder Rootkit on VM

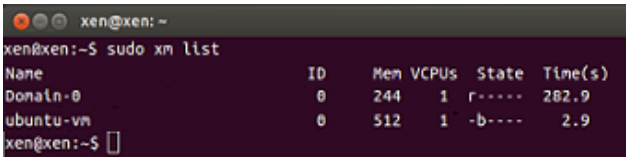

Fig. 6. Virtual Machine Instances Running on Xen Hypervisor

Average Coder [24] is a Linux based Kernel-level rootkit. It uses the LKM to inject the malicious code into the kernel space during run time of the operating system. It has the capability to hide itself from lsmode and also hides process, TCP connections, ports and logged-in users details. We have tested this rootkit by injecting onto a Linux distribution of Ubuntu 12.04 LTS virtual machine. Fig.5 provides a Ubuntu 12.04 LTS virtual machine. Fig.5 provides a screenshot of successful injection of an average coder rootkit. Functionalities of rootkits used in this experiment are depicted in table1.

Table 1. Functionalities of Rootkits

Rootkit	Process	Port	File Hiding	Root Privilege	User Mode	Kernel Mode	VMI Introspected
Jynx2	✓	✓	✓	✓	✓	X	✓
Avarage Coder	✓	✓	✓	X	X	✓	✓

VII. SCOPE AND ASSUMPTIONS

Our proposed System: malicious process detection in guest operating system using VMI assumes that Xen hypervisor is trustworthy. We believe that our proposed work is one new solution by rely on existing methods [12] [13]. Semantic gap of VMI is hard to address and yet to be addressed to make the VMI based approach more realistic and error-free. The use of LibVMI in this work presently focusing virtual machine memory inspection.

VIII. CONCLUSION AND FUTURE WORK

In this paper, we have presented Virtual Machine Introspection based malicious process detection approach for virtual machines. It is based on extracted system call information from introspected virtual machine memory pages during run state. To differentiate benign process from malicious one, system call information is employed. In addition, known opened backdoor ports in conjunction with process information are employed to label malicious process. Average Coder rootkit and Jynx2 rootkit are successfully injected onto a virtual machine to make virtual machine as malicious machine.

Proposed work feasibility evaluation is in progress. As a future work, we planned to measure the detection capability of the proposed work by adopting LibVMI to introspect virtual machine memory content. Introspected virtual machine memory contents does not provide high level information directly due to the raw nature (binary form or hex dump). This semantic gap needs to be addressed to derive meaningful information out of raw data. After the successful evolution of our proposed work, we planned to conduct the suitability test on different hypervisor to scrutiny the compatibility.

REFRENCES

[1] http://www.zdnet.com/hypervisors-are-the-pillars-of-the-cloud-the-cloud-not-the-achilles-heel-7000027931/J

[2] “Agobot,” http://www.f-secure.com/v-descs/agobot.shtml, 2012

[3] T. Garfinkel and M. Rosenblum,.“A Virtual Machine Introspection Based Architecture for Intrusion Detection” In Proceedings of the Symposium on Network and Distributed Systems Security (SNDSS), February 2003, pp 191-206.

[4] Payne, B. D., Carbone, M., and Lee, “Secure and flexible monitoring of virtual machines”.In Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC 2007). Dec. 2007 Miami Beach, FL

[5] Srinivasan, D, Wang, Z, Jiang, X, and Xu, D. “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 (CCS’11). ACM, pp 363–374.

[6] Hay, B. and Nance, K. “Forensics examination of volatile system data using virtual introspection ”. *SIGOPS Operat. Syst. Rev. 42*, pp 74–82.

[7] https://code.google.com/p/xenaccess/

[8] http://www.xenproject.org/

[9] http://www.linux-kvm.org/page/Main_Page

[10] http://libguestfs.org/

[11] http://www.vmware.com/company/news/releases/vmsafe_vmworld

[12] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, W. Lee “Virtuoso: narrowing the semantic gap in virtual machine introspection” IEEE Symposium on Security and Privacy (SP ’11), IEEE Computer Society, Washington, DC, USA (2011), pp. 297–312

[13] FU, Y., AND LIN, Z. “Bridging the semantic gap in virtual machine introspection via online kernel data redirection ”. ACM Trans. Inf. Syst. Secure. 16, 2 (2013)\

[14] http://blogs.bromium.com/2012/10/01/mind-the-gap-the-limitations-of-vm-introspection/

[15] Wesley Vollmar, Thomas Harris, Lowell Long,and Robert Green, “Hypervisor Security in Cloud Computing Systems”, ACM Computing Surveys, Vol.0.No.0, Article 0, Publication Date 2014.

[16] Hyun wook Baek, Abhinav Srivastava, Jacobus Van der Merwe,. “CloudVMI: Virtual Machine Introspection as a Cloud Service” In Proceeding IC2E '14 Proceedings of the 2014 IEEE International Conference on Cloud Engineering Pages 153-158 Washington, DC, USA 2014

[17] Thu Yein Win, Huaglory Tianfield, Quentin Mair, Taimur Al Said, Omer F. Rana “Virtual machine introspection” In Proceedings of the 7th International Conference on Security of Information and Networks September 2014 SIN '14

[18] J. Hizver and T. Chiueh, “Real-time Deep Virtual Machine Introspection and Its Applications,” in Procedings of International conference on Virtual Execution Environments, 2014, pp. 3–14

[19] CHIANG, J.-H., LI, H.-L., AND CHIUEH, T.-C. “Introspection based memory de-duplication and migration”. In Proc. of International Conference on Virtual Execution Environments (VEE) (2013).

[20] https://www.usenix.org/legacy/publications/library/proceedings/osdi02/tech/full_papers/waldspurger/waldspurger_html/node6.html

[21] B. Dolan-Gavitt, B. Payne, and W. Lee, "Leveraging forensic tools for virtual machine introspection," School of Computer Science, Georgia Institute of Technology, Tech. Rep. GT-CS- 11-05, 2011. 22 Hay and Nance, 2008 B. Hay, K. Nance Forensics examination of volatile system data using virtual introspection SIGOPS Oper. Syst. Rev., 42 (3) (2008), pp. 74–82

[22] Packet Storm (Last accessed 14 January 2015), http://packetstormsecurity.org/UNIX/penetration/rootkits/

bdoor.c,blackhole.c,cheetah.c,server.c,ovas0n.c

[23] Haiquan Xiong,Zhiyong Liu1,Weizhi Xu1,,Shuai Jiao1," Libvmi:A Library for Bridging the Semantic Gap between Guest OS and VMM" published in 2012 IEEE 12th International Conference on Computer and Information Technology page no 549 – 556 ,Oct. 2012 Chengdu.

[24] http://average-coder.blogspot.in/2011/12/linux-rootkit.html

[25] http://packetstormsecurity.com/

[26] Ping Chen , Peng Liu , Bing Mao, "System Call Redirection: A Practical Approach to Meeting Real-world Virtual Machine Introspection Needs" In Proceedings of The 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 23-26 June 2014 Atlanta, GA: pp: 574 – 585

[27] Yutao Liu, Yubin Xia , Haibing Guan , Binyu Zang, "Concurrent and consistent virtual machine introspection with hardware transactional memory " in procedings of IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014 5-19 Feb. Orlando, FL ,PP: 416 - 427