2015 IEEE 21st International Conference on Parallel and Distributed Systems

Execution Time Measurement of Virtual Machine Volatile Artifacts Analyzers

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.edu.in

***Abstract*— Due to a rapid revaluation in a virtualization environment, Virtual Machines (VMs) are target point for an attacker to gain privileged access of the virtual infrastructure. The Advanced Persistent Threats (APTs) such as malware, rootkit, spyware, etc. are more potent to bypass the existing defense mechanisms designed for VM. To address this issue, Virtual Machine Introspection (VMI) emerged as a promising approach that monitors run state of the VM externally from hypervisor. However, limitation of VMI lies with semantic gap. An open source tool called LibVMI address the semantic gap. Memory Forensic Analysis (MFA) tool such as *Volatility* can also be used to address the semantic gap. But, it needs to capture a memory dump (RAM) as input. Memory dump acquires time and its analysis time is highly crucial if Intrusion Detection System IDS (IDS) depends on the data supplied by FAM or *VMI* tool. In this work, live virtual machine RAM dump acquire time of *LibVMI* is measured. In addition, captured memory dump analysis time consumed by Volatility is measured and compared with other memory analyzer such as *Rekall*. It is observed through experimental results that, *Rekall* takes more execution time as compared to Volatility for most of the plugins. Further, Volatility and *Rekall* are compared with LibVMI. It is noticed that examining the volatile data through LibVMI is faster as it eliminates memory dump acquire time.**

Keywords—Hypervisor, Semantic gap, Intrusion Detection System, Memory Forensic Analysis, Rootkit, Virtual Machine Introspection.

I. INTRODUCTION

Virtualization technology has sprawl rapidly over the last few years and it has been one of the most potent forces for reshaping the traditional landscape of the computing devices such as servers, desktop, networks, etc., Virtualization facilitates sharing of physical computing resources among different guest virtual machines by using special software layer called hypervisor or Virtual Machine Monitor (VMM). Virtualization platform becoming an attractive target for an adversary due to easily accessible of virtual machines through Cloud Service Provider (CSP) [1]. An adversary could design and run a rootkit or malware that could alter the normal behavior of the legitimate guest operating system either by modifying System Call Table (SCT) or Interrupt Descriptor Table (IDT) or some other critical operating system data structure [2]. Later such malignant software can acquire the control of virtual machine by evading existing defense mechanism, e.g. anti-virus or agent based solution. Protecting virtual machines from advanced, sophisticated malware or threats is a highly exigent task for CSP.

The current generation of host-based, anti-malware prevention systems are agent based, signature dependent and they run inside the host machines. They are inadequate to thwart against emerging advanced malware attack[17]. Similarly, they are ineffective for virtual environment as their functionalities restricted only to a single system. In a virtualized environment, hypervisor is able to manage the multiple guest operating system[1]. Protecting individual guest OS by placing Host based Intrusion Detection System (HIDS) or anti-malware solution is ineffective. To overcome this problem,Virtual Machine Introspection (VMI) [3],[4],[8],[9] has emerged as fine-grained technique that provides complete visibility of run state of the virtual machines at hypervisor. The process of viewing the run state of the virtual machine from hypervisor is named as VMI[3]. The main motivation behind VMI is to scrutinize any abnormal change occurs during run state of the virtual machine. Monitoring true state of the virtual machine without compromising the performance as well as without the knowledge of virtual machine is an active research topic. Many proposed solutions have adopted VMI to identify malignant activities of the virtual machine [6],[5][18]. An open source VMI tool called LibVMI [7] is able to provide run state of the live VM and also capable to acquire live VM RAM dump.

RAM dump capture time and its analysis time in real time are highly crucial if an IDS depends on data supplied by MFA tool or VMI tool. Furthermore, memory analyzer accuracy is also a primary for IDS. Thus, our aim is to 1). Measure the time required to capture a live VM RAM dump using VMI tool. 2). Measure the performance of the MFA tool such as *Volatility*[12] in terms of execution time elapsed to analyze the RAM dump of different size. 3). Compare the performance of the *Volatility* with another open source MFA tool called *Rekall*[14] in terms of execution speed. 4). Inject real world rootkits onto the virtual machine in real time, view the internal shape of the VM using VMI tool, capture the RAM dump and analyze them using *Volatility* and *Rekall* separately to appraise the detection accuracy.

The rest of the paper is structured as follows: Section II provides background. Section III discusses related work. Section IV describes the motivation to carry out this work. Evaluation and experimental results are discussed in section V

1521-9097/15 $31.00 © 2015 IEEE
DOI 10.1109/ICPADS.2015.47

and finally conclusion and future work presented in section VI.

II . BACKGROUND

A. *Semantic Gap of Virtual Machine Introspection*

The VMI has emerged as state-of-the-art technique as it is able to provide the internal behavior of the target virtual machine by examining volatile data [3][10]. As a result, it helps to spot quickly the abnormality of the system due to malware or intrusions. Run state of the virtual machine is able to view externally using VMI. However, obtained state information is in binary form 0's and 1's i.e., in raw form. Extraction of high level information such as an active process list, system calls, module list and network connections, etc., from raw data are referred as semantic gap of VMI [5],[6],[18],[25] and it is the biggest challenge as it needs knowledge of the guest operating system [6][24]. Fig.1.depicts an overview of semantic gap. Internal view represents virtual machine level observations, whereas the external view denotes inspections at VMM. Strong isolation property of VMI resists malware or threats.[11][16].

B. *Rootkits*

The Rootkits are a special type of malware that runs at the highest privilege access of operating system (kernel ring '0') by exploiting the security weakness present on the system. They maintain a persistent and undetectable presence on the victim system by hiding most privileged access to OS utility. The rootkits deviates normal behavior of the system by injecting malicious code into an operating system. A family of rootkits is much sophosticated to compromise the OS kernel by performing DKSM (Direct Kernel Structure Manipulation) attack [22][23]. They have the capacity to foil or bypass In-the-Box solution and the rootkits are often directly alter the kernel memory using the */dev/kmem* memory device. Moreover, they also temper raw of kernel data structure. Similarly, some rootkits are more potent to accomplish Semantic Value Manipulation (SVM) attack [22] by manipulating semantic data value of important kernel data structure.

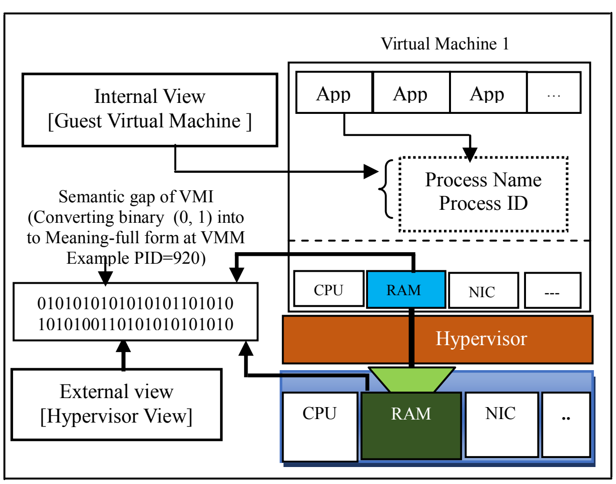

Fig. 1. Semantic Gap of Virtual Machine Introspection

C. Virtual Machine Introspection Tool: LibVMI

The LibVMI is a library that bridges the semantic gap between VM and VMM [7][28] and it is an extended version based on XenAccess Library**.** By operating at hypervisor, it is able to provide run state of the target virtual machine information like running processes list, module list, event details. However, it needs few Guest Operating System (GOS) details prior to introspection. For Linux GOS, it needs *ostype, sysmap, linux_tasks, linux_mm, linux_pid, linux_pgd*. Similarly, it needs information such as *ostype, sysmap, win_tasks*, *win_pdbase* and *win_pid* for windows GOS. The *LibVMI* also supports to capture the RAM dump of the target virtual machine. MFA tool is needed to obtain run state of the VM from RAM dump.

D. *Memory Forensic Analysis Tool*

MFA tools can also be used to address the semantic gap of VMI[13]. A popular MFA opens source tool called *Volatility*[12] is intelligent enough to analyze the RAM dump. However, it is unable to capture the memory dump and it is not an IDS. Thus, memory dump is required to capture before the analysis. Intrusion detection is achieved only after careful investigation of relevant fields present in the memory dump. Similarly, *Rekall*[14] is another advanced open source memory forensic framework historically forks with *Volatility* with enhanced optimized performance. Both *Volatility* and *Rekall* are matured enough to analyze Linux system memory dump as well as a Windows system memory dump.

III. RELATED WORK

VM Introspection Based: The idea of VMI invented by Garfinkel and Rosenblum [3] to detect intrusions during run state of virtual machines by positioning IDS at the hypervisor. This idea was implemented using the prototype Livewire [3]. Afterwards a number of solutions have proposed in this direction for different aspects including malware detection [15][16], memory forensics [10][21] etc.

The key challenge of VMI is semantic gap. The Virtuoso [6] is one such approach to bridge the semantic gap of VMI by automatically creating introspection tool that can semantically extract meaningful information of guest virtual machine based on low level data source. This is achieved by using dynamic slicing algorithm. The main limitation of this technique is that it is not fully automated and it requires minimal human effort. The same approach extended and proposed with a name VMST [5]. It addresses 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 i.e., involvement from end users. The Virtuoso and VMST created more attention to build a semantic gap, but they still limited to satisfy usefulness and practicality of VMI[19]. Moreover, these techniques have a high overhead to address the semantic gap when different versions of the guest operating system offered by CSP. To address this issue, “shadow context” based approach has proposed to meet real world needs of VMI by significantly improving the practical usefulness of VMI [19]. It encourages one inspection program to inspect different version of the guest virtual machine.

Memory Forensic Analysis (MFA) Based: forensic memory community grappled VMI gap to analyze forensically critical kernel relevant information from dump of physical memory[13]. VMI based approach called Virtual Introspection for XEN (VIX) captures the volatile data of the XEN virtual machine [10] by pausing the target virtual machine till the completion of acquisition then resumes the target virtual machine. Suspension of target virtual machine functions during the data acquisition process ensures the integrity of the state information. In the same direction, few more work [20], [21] has exposed the raw low level-byte artifacts of guest VM memory to investigate relevant kernel data structure alteration based on memory dump. The prototype called MOSS[2][26] developed to extract the complete semantic view of the kernel data structure that are altered by DKSM/SVM attack.

IV. MOTIVATION

Once the kernel level rootkit or advanced persistent malware penetrate into the core of operating system kernel, they can change the behavior of the legitimate system by arbitrarily modifying the SCT or IDT or any other critical kernel code and data structure. It is very difficult to detect such changes if they occurs during run time of the system. Some advanced rootkits or malware competent enough to bypass or disable the anti-malware/HIDS to evade the detection. One best solution to catch abnormality of the system is by analyzing the RAM contents as it provides accurate state of the system. For example, process list, loaded driver modules, SCT , IDT details ect.

In a virtualized environment, one of the best ways of examining the RAM contents of the virtual machine is via VMI tool. For example LibVMI is proficient to provide the internal shape of the target virtual machine like the active process list, module list, event details, etc. In addition, it supports to capture the RAM dump. However, LibVMI is not rich enough to provide more kernel information due to its limited functionalities as on today. The sophisticated DKOM and SVM attacks based rootkits are more potent to alter the guest kernel data structure. To view in-depth semantic information of the virtual machine, another way is by analyzing the RAM dump using MFA tool.

Without IDS, only viewing the internal state of the virtual machine either through VMI tool or MFA tool is inadequate to classify the system state is normal or abnormal. Furthermore, without IDS, it is impractical to safeguard the virtualized environment against malware attack or other types of attacks. The prime requirement to safeguard the virtualized environment round the clock is IDS. However, HIDS is an ineffective solution for virtualized environments to thwart advanced malware attacks. Thus, our proposed architectural Hypervisor based Intrusion Detection System (HyIDS) framework is the supreme solution to uncover abnormality of the virtual machine by inspecting volatile data. The HyIDS needs state of the virtual machine to classify the system state as normal or abnormal. If HyIDS depends on the data supplied by the VMI tool or MFA tool, the time needed to fetch the state of the system by the VMI tool or MFA tool plays an important role. Fig. 2 shows the high level architectural

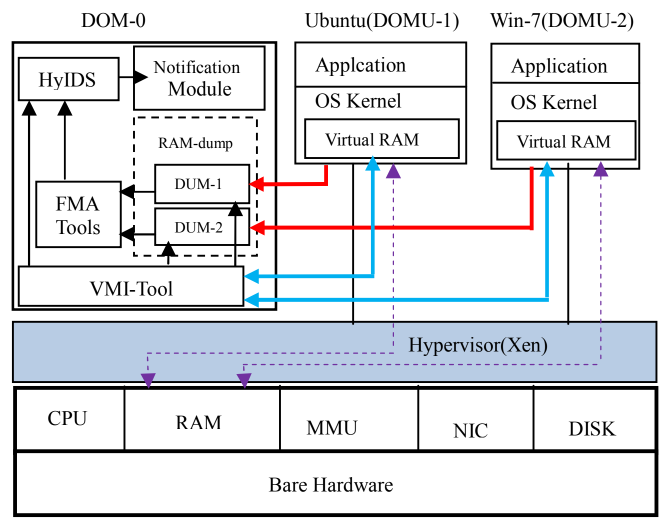

Fig. 2. High level view of HyIDS Architecture on Virtualized Environment

framework of HyIDS based virtualized environment. The HyIDS receives true state of the virtual machine either by MFA tool or VMI tool. In this scenario, the time required to fetch the real state of the virtual machine by reading volatile data is highly crucial. With this motive in this work, we have measured and compared the execution time of *Volatility* with *Rekall*. Further, speed of *LibVMI* is compared with *Volatility* and *Rekall*. Finally, we figure out which is the best feasible solution to secure the virtulized environment while addressing so called semantic gap.

V. EVALUATION AND EXPERIMENTAL RESULTS

We have evaluated the execution time of *Volatility* and *Rekall* for the different RAM dump size of 1GB, 2GB and 3GB. Memory dump of both Linux virtual machine and windows virtual machine captured using open source VMI tool called LibVMI.

A. Experimental Setup

The experiments conducted on the host system which posses the following specification: Intel (R) core(TM) i7-3770 CPU@ 3.40GHz, 20GB RAM, Ubuntu 14.04 (Trusty tahr) (64-bit) operating system. The popular open source Xen 4.4 bare metal hypervisor had utilized to establish the virtualized environment. To introspect and forensically investigate the run state of the live virtual machine memory, we have created two guest virtual machines of different operating system such as *Ubuntu 12.04.3-LTS* as *DOMU-1* and *Windows-7SP0-64x* as *DOMU-2* under Xen hypervisor. Both of them are managed by DOM-0 management unit. Popular introspection tools such as LibVMI version 0.10.1. installed on the most privileged domain (DOM-0) of Xen hypervisor to introspect low-level artifact's of the target virtual machine as well as to capture the live RAM dump. LibVMI trap the hardware events and access the vCPU registers while functioning at hypervisor. MFA tools such as *Volatility* version 2.4 and *Rekal* version 1.3.2 (dammastack) employed to examine the captured RAM dump.

B. *Virtual Machine RAM Dump Analysis using Volatility and Rekall*

Volatility is one of the most widely used open source memory forensic tools used to extract digital artefact's from volatile memory (RAM) samples. It offers a vast number of built-in plug-ins to investigating different operating system memory dump. This makes the *Volatility* to use extensively as first choice for digital investigation of RAM samples. Operating kernel data structure details are used during analysis time and this detail made available to *Volatility* through the profile. Windows operating system profiles are inbuilt including recent windows-8-1. In case of Linux based operating systems, *Volatility* requires user to create the profile of respective Linux distribution before the RAM dump analysis. This is due to continuous Linux kernel version consistently updating.

We have a created profile for Ubuntu 12.04 virtual machine and used the same profile during the experiments. Live Ubuntu 12.04 virtual machine RAM dump of size 1GB, 2GB and 3GB has acquired using *LibVMI*. The captured RAM dumps have analyzed using the *Volatility* Linux plugins such *as Linux_pslist, Linux_lsmode, Linux_arp, Linux_check_idt, Linux_cpuinfo, Linux_dmesg, Linux_iomem, Linux_lsof, Linux_netstat, Linux_psaux, Linux_pslist_calhe, Linux_pstree, Linux_pstree*. The same RAM dumps have also analyzed by another memory analyzer called *Rekall.* Linux plugins name of *Rekall* are as same as Volatility Linux plugins name.

We have compared *Volatility* Linux plugins execution time with *Reakll* Linux plugins execution time to evaluate the performance in terms of processing time. Fig 3, 4 and 5 depict execution time taken by *Volatility* and *Rekall* for 1GB, 2GB and 3GB RAM dump respectively. From the experimental results, it is observed that *Rekall* execution time is more for the following plugins *Linux_pslist, Linux_lsmode, Linux_arp, Linux_cpuinfo, Linux_dmesg, Linux_iomem, Linux_netstat, Linux_psaux, Linux_pslist_calhe, Linux_pstree,* as compared to *Volatality*. However, *Rekall* processing time is faster for *Linux_check_idt, Linux_lsof, Linux_psview* plugins as compared to *Volatility*.

Some of the most common windows plugins of *Volatility* and *Rekall* are tested on Windows-7 virtual machine memory dump of size 1GB, 2GB and 3GB were used to conduct the experiments. Fig 6, 7 and 8 depicts an execution time taken by *Volatility* and *Rekall* for 1GB, 2GB and 3GB RAM dump respectively. Our experimental results demonstrate that Rekall takes more time to execute the following plugins *pslist, dlllist, eventhooks, handles*, *ldmodules, malfind,* modules, *multiscan, netscan, psscan, pstree, ssdt* compared to *Volatility*.

Another major observation, we found that Volatility reported time for the following plugins *Linu_Syscall* (110s), *Linux_lsof* (85s) and *Linux_mem*(88s) is high compared to other plugins. But, for the same plugins, execution time is drastically reduced in Rekall.

C. Detecting Kernel Level Rootkits

To evaluate trustworthiness of live Virtual Machine Introspection and memory forensic tool, we have injected publically available [27] real world rootkis on both Windows and Ubuntu guest VM. We have used seven linux kernel level rootkit such as *Simplerootkit[SR], Kbeast[KB],chkrootkit-0.50[CK], avarage coder[AC], adore-ng[Adr-ng],open-hijack[OH],getpid-hijack[G H]* Windows operating system based kernel rootkits called *FU-rootkit [FU] and Hacker Defender[HD]* injected onto Windows-7 virtual machine. Table1 provides rootkits explored in this work with the guest operating system on which they were injected. We practilaly explored that, the LibVMI is capable to detect injected rootkits (malicious process ID, hidden modules, etc) on the live running virtual machine. A more semantic information extracted by MFA tools such as Volatility and Rekall on the captured RAM dump of both Windows and Ubuntu.

As a first step of rooktit detection, true run state of the VM viewed using *module-list* plugin of LibVMI while working at hypervisor (DOM-0). As a proof of experimental results, we have mentioned a live snapshot of average coder rootkit in the Fig.9. Injected rootkit module successfully detected by *LibVMI* whereas the same module was unable to view against inspection carried at the virtual machine through *lsmod* command. In Fig.9. The background GUI screenshot on the right side shows the output of *module-list* plugin of LibVM in which inserted rootkit module "*rootkit"* is visible whereas same rootkit module is completely hidden against the inspection executed at the infected virtual machine (DOMU-1) through *lsmod* utility see foreground screenshot on the left side The figure 10 and figure 11 presents the output of *Linux-lsmod* plugin of *Volatility* and *Rekall* respectively.

Table1: Real World Rootkit Expriment under Guest Virtual Machines

Rootkits	OS	Functionalities	Behavior
SR, AC, KB	Ubuntu 12.04	lsmod, sys-call,ps, hf.	DKSM/SVM
CK,AD-ng,	Ubuntu 12.04	Sys-call,ps, mod, strg	DKSM/SVM
OP, GH	Ubuntu 12.04	Sys-call,PID-hijak, ps,	DKSM/SVM
HD, FU	Windows-7	Sys-call-hijak, ps, fl	DKSM/SVM

SR: Simple Rootkit, **AC:** Avarage coder, **KB:** Kbeast, **CK**: Chkrootkit 0.50 **OP:**open-hijack **GH**: getpid-hijack, **HD:** Hacker Defender, **FU**-Fu rootkit, AD-ng-Adore-ng

Table 2: RAM Dump Analysis Time of UBUNTU Guest Virtual Machine

RAM Dump Size	LibVMI Ubuntu 12.04 (GOS)		Volatility Ubuntu 12.04 (GOS)		Rekall Ubuntu 12.04 (GOS)	
	Process List	Module List	Process List	Module List	Process List	Module List
1GB	0.30s	0.22s	3.31s	3.69s	5.10s	4.19s
2GB	0.32s	0.29s	3.24s	3.85s	5.85s	4.89s
3GB	0.34s	0.34s	3.98s	4.12s	7.85s	5.01s

Table 3: RAM Dump Analysis Time of WINDOWS Guest Virtual Machine

RAM Dump Size	LibVMI Windows-7 (GOS)		Volatility Windows-7 (GOS)		Rekall Windows-7 (GOS)	
	Process list	Module list	Process list	Module list	Process list	Module list
1GB	0.32s	0.26s	2.25s	2.23s	8.76s	2.92s
2GB	0.38s	0.45s	2.58s	2.64s	10.97s	3.07s
3GB	0.41s	0.58s	2.68s	3.29s	11.8s	7.84s

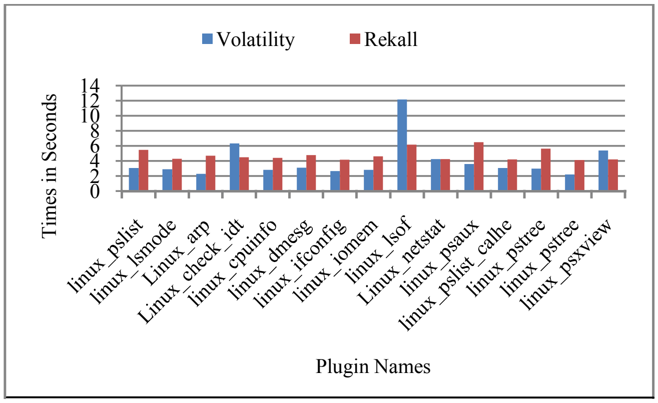

Fig. 3. Analysis of Ubuntu 12.04 VM - 1GB RAM Dump

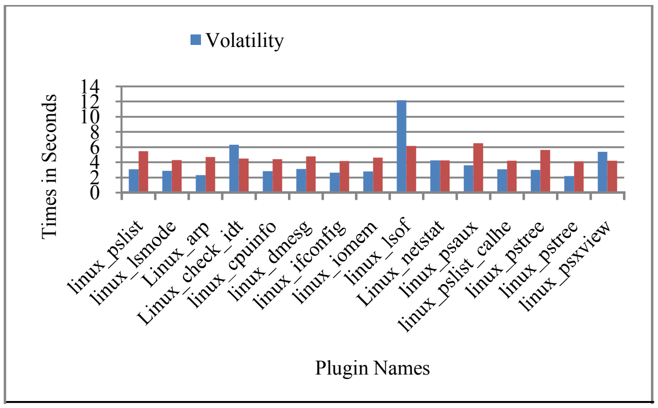

Fig. 4. Analysis of Ubuntu 12.04 VM - 2GB RAM Dump

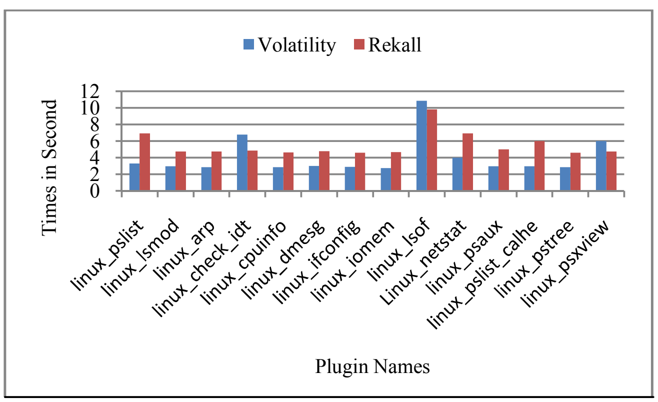

Fig. 5. Analysis of Ubuntu 12.04 VM - 3GB RAM Dump

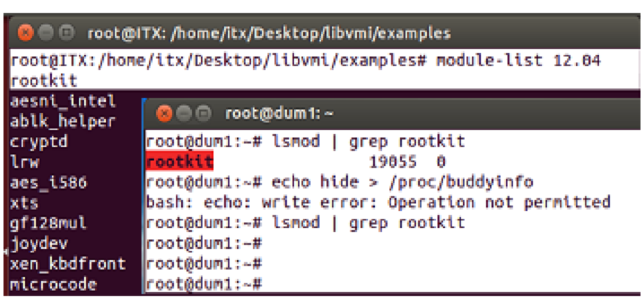

Fig.9. AC Rootkit module hidden by an attacker at DOMU-1 VM the same detected by out-of-the-box VMI solution LibVMI

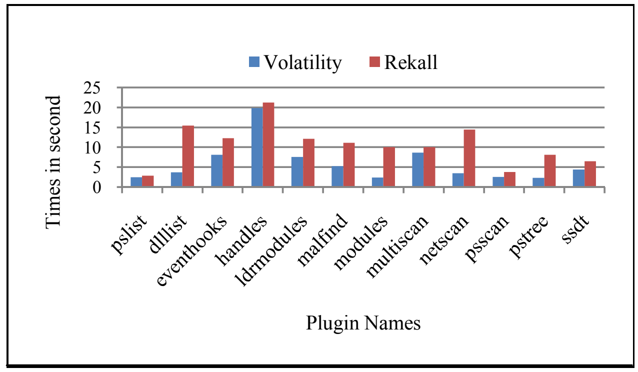

Fig. 6. Analysis of Windows-7SP0- 1GB RAM Dump

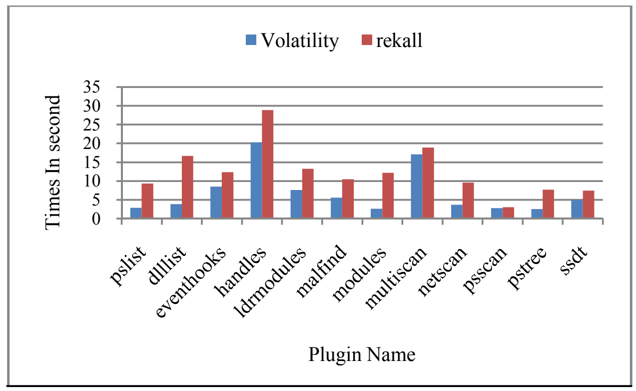

Fig. 7. Analysis of Windows-7SP0- 2GB RAM Dump

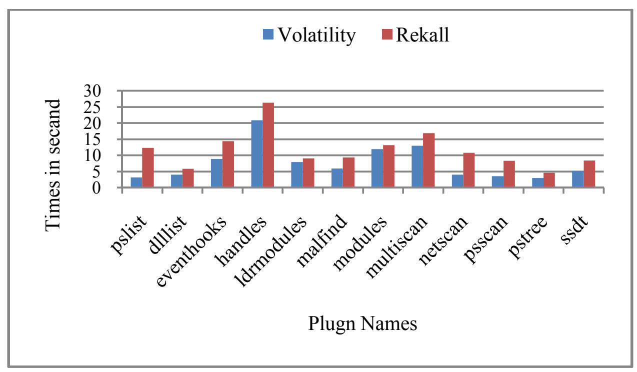

Fig. 8. Analysis of Windows-7SP0- 3GB RAM Dump

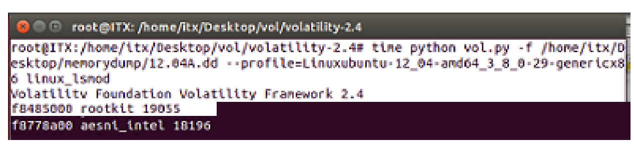

Fig.10. AC Rootkit hidden module extracted by Volatility from raw of physical memory dump

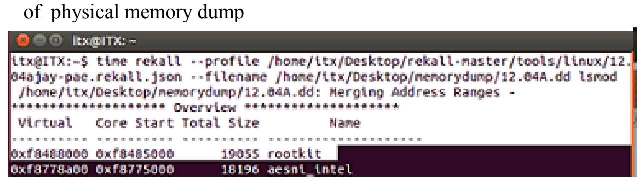

Fig.11. AC Rootkit infected module extracted by Rekall from raw of physical memory dump

From the figures 10 and 11, we can observe that both *Volatility* and *Rekall* are capable to report correctly the hidden kernel module of average coder "*rootkit"* from RAM dump. The extraction speed of *LibVMI*, *Volatility* and *Rekall* for *pslist* and *module-list* plugins is tabulated in table 2 and table 3. We can observe that LibVMI fetching speed is faster as compared to *Volatility* and *Rekall*.

VI. CONCLUSION AND FUTURE WORK

One way to spot malicious activities of the virtual machine is through viewing run state of the live virtual machine using LibVMI. Alternate way is by analyzing RAM dump of the virtual machine using MFA tool. In this work, the execution speed of *Volatility* is measured and compared with *Rekall*. It is noticed that the *Rekall* execution speed is slow for most of the plugins as compared to *Volatility*. Both *Volatility* and *Rekall* are capable to address the semantic gap by providing readable information from RAM dump. However, they need memory dump to initiate the analysis.

The live virtual machine state information extraction through *Volatility* and *Reka*ll is slower as compared to *LibVMI.* However, *LibVMI* is not matured enough to provide more semantic state information. In other words, currently *LibVMI* possessing limited to few plugins. As there is no memory dump acquire time involved in *VMI* based approach (*LibVMI*), speed of retrieving the data from volatile memory is faster as compared to memory dump based approach (*Volatility* and *Rekall*). In this context, the *hyIDS* get state information quickly, which helps in determining the intrusions rapidly. As future work, we plan to develop more program module for an existing LibVMI tool to detect intrusions or malware that strengthen virtualized environment.

References

[1]. Wesley Vollmar, Thomas Harris, Lowell Long,and Robert Green 2013, "Hypervisor Security in Cloud Computing Systems" , ACM Computing Surveys, Vol.0.No.0, Article 0, Publication Date 2014.

[2]. Aravind Prakash , Eknath Venkataramani , Heng Yin , Zhiqiang Lin, "Manipulating semantic values in kernel data structures: Attack assessments and implications", Proceedings of the 2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), p.1-12, June 24-27, 2013

[3]. T. Garfinkel and M. Rosenblum., "A Virtual Machine Introspection Based Architecture for Intrusion Detection", Proceedings of the Symposium on Network and Distributed Systems Security (SNDSS), pages191-206, February 2003

[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).

[5]. FU, Y., AND LIN, Z. "Bridging the semantic gap in virtual machine introspection via online kernel data redirection". ACM Transaction on information Security. 16, 2 , 2013

[6]. 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

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

[8]. Florian Westphal , Stefan Axelsson , Christian Neuhaus , Andreas Polze," VMI-PL: A monitoring language for virtual platforms using virtual machine introspection", Digital Investigation: The International Journal of Digital Forensics & Incident Response, 11, p.S85-S94, August, 2014

[9]. D. Srinivasan, Z. Wang, X. Jiang, and D. Xu, "Process out-grafting: an efficient "out-of-vm" approach for fine-grained process execution monitoring," in Proc. of the 18th ACM conference on Computer and communications security (CCS'11), October 2011.

[10]. B. Hay, K. Nance, "Forensic examination of volatile system data using virtual introspection", ACM SIGOPS Operating Systems Review, vol. 42(3) ISSN: 0163-5980 April 2008.

[11]. Xuxian Jiang , Xinyuan Wang , Dongyan Xu, "Stealthy malware detection through vmm-based "out-of-the-box" semantic view reconstruction", Proceedings of the 14th ACM conference on Computer and communications security, November 02-October 31, 2007, Alexandria, Virginia, USA

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

[13]. B. Dolan-Gavitt, B. Payne, and W. Lee, "Leveraging forensic tools for virtual machine introspection", Technical Report; GT-CS-11-05, 2011.

[14]. http://www.rekall-forensic.com/

[15]. Jiang, X., Wang, X., and Xu, D. 2007. "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 (CCS'07)*. ACM, 128–138.

[16]. Dinaburg, A., Royal, P., Sharif, M., and Lee, W. 2008. "Ether: Malware analysis via hardware virtualization extensions". In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, 51–62.

[17]. Kumara.M.A Ajay, Jaidhar C.D "Hypervisor and Virtual Machine Dependent Intrusion Detection and Prevention System for Virtualized Cloud Envoirnoment" 1st IEEE International Conference on Telematics and Future Generation Networks (TAFGEN2015) May 26-28, 2015 Kuala Lumpur, Malaysia

[18]. Yangchun Fu , Zhiqiang Lin, EXTERIOR: using a dual-VM based external shell for guest-OS introspection, configuration, and recovery, Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, March 16-17, 2013, Houston, Texas, USA

[19]. R. Wu, P. Chen, P. Liu, and B. Mao, "System Call Redirection: A Practical Approach to Meeting Real-world Virtual Machine Introspection Needs," in *DSN*, Jun. 2014, pp. 1--12.

[20]. S. Garfinkel, P. Farrella, V. Roussev and G. Dinolt, (2009), "Bringing science to digital forensics with standardised forensic corpora", Digital Investigation 6 S2-S11.

[21]. A. Case, A. Cristina, L. Marziale, G. Richard and V. Roussev, "FACE: Automated Digital Evidence Discovery and Correlation", in Proceedings of the Eighth Annual DFRWS Conference, 2008

[22]. Aravind Prakash, Eknath Venkataramani, Heng Yin and Zhiqiang Lin. "On the Trustworthiness of Memory Analysis —An Empirical Study from the Perspective of Binary Execution" *In IEEE Transactions on Dependable and Secure Computing (**TDSC**)*, 2014

[23]. S. Bahram, X. Jiang, Z. Wang, M. Grace, J. Li, D. Srinivasan, J. Rhee, and D. Xu. "DKSM: Subverting virtual machine introspection for fun and profit". IEEE Symposium on Reliable Distributed Systems, 2010.

[24]. Bach, M. J. 1986. *The Design of the UNIX Operating System*. Prentice Hall.

[25]. Peter M. Chen , Brian D. Noble, When Virtual Is Better Than Real, Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, p.133, May 20-22, 2001

[26]. Q. Feng, A. Prakash, H. Yin, and Z. Lin." Mace: High-coverage and robust memory analysis for commodity operating systems", in Proceedings of ACSAC'14 30th Annual Computer Security Applications Conference

[27]. https://packetstormsecurity.com/

[28]. B. D. Payne. Simplifying Virtual Machine Introspection Using LibVMI. Sandia Report, 2012.