ISSN 1870-4069

A Method for Malware Analysis by Virtual Machine Introspection Technique

Luis Enrique Héctor Almaraz García, Raúl Acosta Bermejo

Instituto Politécnico Nacional, Centro de Investigación en Computación, Mexico City, Mexico
`lalmarazg1400@alumno.ipn.mx, racostab@ cic.ipn.mx`

Abstract. Malicious code has become one of the biggest threats in the field of computer security. Traditional malware monitoring tools are installed in the physical host, they trust in the integrity of the host, however, they are vulnerable to being infected by malware and delivering erroneous results about monitoring. In this paper, a method based on Virtual Machine Introspection technique is proposed to obtain the memory image of a Virtual Machine, from outside, with the help of the VirtualBox API, also analyze its internal content such as running processes, threads, network connections, and open files with the use of the Volatility Framework to interpret the low-level bytes into high-level information and finally, report this information in a monitoring register. This approach has been tested with the execution of 3 samples of malware inside a 32-bit Microsoft Windows XP SP3 Virtual Machine and the results obtained support the main hypothesis that if the Virtual Machine Introspection technique is applied to a Virtual Machine then it is possible to obtain the activities of a process and according to its behavior, identify malware.

Keywords: virtual machine introspection, malware, process monitoring, memory forensics, dynamic analysis.

1 Introduction

Computer forensics is the science that identifies, collects, preserves, analyses and presents the data that has been processed and stored on electronic media, in a legal process [1]. Virtual Machines provides efficient use of resources, ease of management, low operation and maintenance costs, flexible systems and even efficient power consumption [2, 3]. Virtual Machine forensic has focused on collecting forensic data to uncover the malicious content in the memory [4, 5], in this case, malware.

Malware or malicious code refers to a program that is inserted into another program and that compromises the confidentiality, integrity or availability of data, applications or the operating system itself [6]. Malicious code has become one of the biggest threats in the field of computer security, the number of malware has grown in recent years [7], [8] and it is reported that every 4.6 seconds a new malware specimen emerged in 2017 [9].

Traditional malware monitoring tools are installed in the physical host, they trust in the integrity of the host, however, they are vulnerable to being infected by malware and delivering erroneous results about monitoring [10]. The approach proposed in this paper is based on Virtual Machine Introspection (VMI) [11] technique to obtain the memory image of a Virtual Machine (VM), from outside, in this case, with the help of the VirtualBox API [12], analyze its running processes, threads, network connections, and open files with the use of the Volatility Framework [13] and finally, report this information in a monitoring register.

The rest of the paper is organized as follows. Section 2 introduces a background information about the Virtual Machine Introspection technique. Section 3 presents related work to Virtual Machine Introspection in search of possible malicious processes. The design of the proposed method is described in section 4. Section 5 specifies the implementation of the method. The experimental results are provided in section 6 and finally, the conclusions and future work are discussed in section 7.

2 Background

2.1 Virtual Machine Monitor

Virtual Machine Monitor (VMM) or hypervisor is a software that enables communication between Virtual Machines and real host [4]. It provides the virtual environment by means of a Virtual Machine where other programs can be executed just as they do in a real environment [14]. There are two types of VMMs [2]. Type I VMM, Fig. 1, is one that runs directly on the hardware, some examples of hypervisors of this kind are: VMware ESX/ESXi, Citrix Xen Server and Oracle VM, they are used in data centers and in server environment. The type II VMM, Fig. 2, is installed on the operating system of the real host as another user program, examples of this kind of hypervisor are: VMWare Workstation/Fusion/Player, VirtualBox, Parallels and Microsoft Virtual PC.

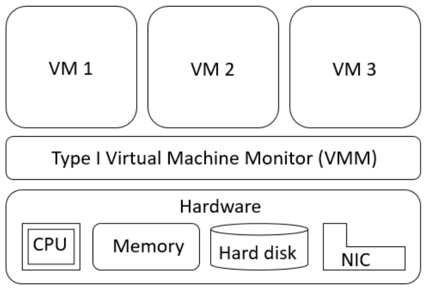

Fig. 1. Type I Virtual Machine Monitor.

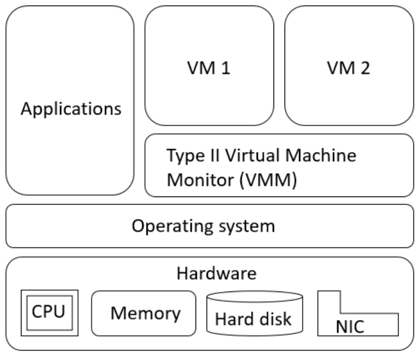

Fig. 2. Type II Virtual Machine Monitor.

2.2 VirtualBox Hypervisor

The approach proposed is based in one of the type II VMM that is called VirtualBox, Fig. 3. It provides a main API that is implemented using the Component Object Model (COM), an interprocess mechanism for software components originally introduced by Microsoft for Microsoft Windows. In this way, in a host with a Microsoft operating system, VirtualBox uses COM and in the rest of operating systems such as Linux and macOS, VirtualBox uses the Cross Platform Component Object Model (XPCOM), a free software implementation of COM originally created by the Mozilla project for their browsers [12]. The VirtualBox front-ends use COM/XPCOM to call the Main API and each VM is working with a VirtualBox client, which helps the VM interact with the VBoxSVC process, the service corresponding to the VMM [14].

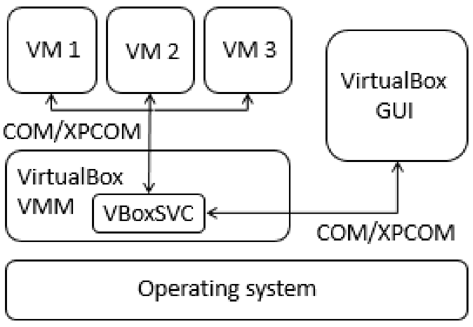

Fig. 3. Architecture of VirtualBox.

2.3 Virtual Machine Introspection

Virtual Machine Introspection is a technique to analyze the memory of a given VM to detect its internal activities from outside over the Virtual Machine Monitor layer [10], [11]. Such activities are related with memory, disk, CPU registers, network connections and available kernel symbols of the VM [15, 16]. This technique has been used for intrusion detection, malware analysis and memory forensics [17]. The method described in this paper performs Virtual Machine Introspection through of the VirtualBox API.

2.4 Volatility Framework

The Volatility Framework is a collection of tools developed in Python, for the extraction of digital artifacts from volatile memory (RAM). The framework is not a memory acquisition tool [13]. It supports investigations for different guest operating systems like macOS, Linux and Windows and extracts the operating system data structure from memory pages of introspected Virtual Machine identifying processes in execution and details of hidden processes that allow detecting malware in the virtual environment [15].

3 Related Work

The proposal of VMI has been introduced since 2013 [18]. In recent years, researchers have adopted this technique to detect malware. Authors in [19] have designed and implemented a process detection system called VmRecoverySystem, with KVM as a hypervisor that consists of four modules. Semantic reconstruction module reconstructs the processes and kernel data structures found in memory. Detection module checks the existence of critical tasks and their system calls in the Virtual Machine. The policy module contains the user's configuration to delete or start a process and likewise, to restart or restore a Virtual Machine.

The recovery module executes the previously actions. They wrote a simple rootkit which tampered the system calls table to test the operation of their system. This idea is also used in [5], with the same KVM hypervisor, but with a scheme of 3 modules, the Virtual Machine process search module searches all the running process identifiers associated with the Virtual Machine. Then, using the Virtual Machine memory dump module to dump the memory of each process obtained previously to finally, the memory forensics analysis module reads the Virtual Machine memory dump file, of each process, to obtain the user, pid, cpu usage and memory usage. Their purpose is to ensure the integrity and efficiency of the information of the processes in the different files.

In [20], a VM Introspection method is introduced to monitor the presence of malware in the volatile memory of the Virtual Machine through the analysis of its processes, files, registers and network activities. They perform introspection to the Xen hypervisor with the help of LibVMI library and analyze system behavior using memory forensics integrating the Volatility Framework. They tested their method running 20 advanced and 8 script-based malware samples and their results were verified with three sandboxes: sandbox CIA, Cuckoo Sandbox and CaptureBAT, they got similar detections in terms of processes, files, registers and network activities. Another approach for Xen hypervisor based Virtual Machine is presented in [15], it has 4 components, the State Information Extraction uses the LibVMI library to reconstruct the memory from raw data to readable information.

The Anomaly based Detection Engine knows which processes access certain system calls, as well as the order of them. The Malicious Port Detection Engine holds a database of well-known backdoor ports. The Notification generates an alert to the user to indicate the presence of possible malware in the Virtual Machine. Average Coder and Jynx2 rootkits were used in their experiment to detect process abnormal behavior through system calls and ports found.

Researchers in [4] propose a mechanism to monitor processes in a VMware Virtual Machine with Windows XP. A snapshot of the Virtual Machine is made. Next, the Virtual Machine is suspended and cryptographic MD5 sums of the next files are taken: vmem, vmsn, and vmss, they keep the information about the current state, snapshots and saved state of the Virtual Machine respectively. They tested their methodology over the Virtual Machine and got the list of processes from the previously files and from the memory dump of the host operating system.

4 Design of the Method

This section details the design and operation of the proposed method to analyze malware in a Virtual Machine through Virtual Machine Introspection technique, see Fig. 4. It consists of the following five steps:

1. Access to the asset: This is achieved through the interprocess mechanism called COM/XPCOM that implements the VirtualBox API.
2. Collection: It generates a memory dump of the Virtual Machine volatile memory.
3. Analysis: It translates the low-level bytes into high-level information with the help of the Volatility tool, through the profile of the virtual machine and extracts objects from the operating system.
4. Logging: It generates the log of the malware analysis.
5. Containment: The COM/XPCOM interprocess mechanism sends a killing command to finish the malware execution from outside the Virtual Machine.

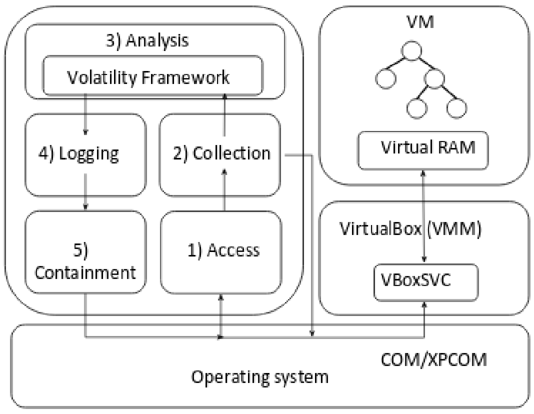

Fig. 4. Architecture of the proposed approach.

5 Implementation of the Method

5.1 Access to the Asset

The first step consists in a programmed module that identifies the Virtual Machine to be monitored from a set of Virtual Machines available from the VirtualBox hypervisor. This is done through the COM/XPCOM interprocess mechanism that calls the VirtualBox API, which in turns, interacts with the VBoxSVC process that manages the current Virtual Machine. This module is responsible for determining the current state of the Virtual Machine, if it is off, then it is turned on and a session is created to interact with the Virtual Machine. If the Virtual Machine state is on from the beginning, then only the session is created.

5.2 Collection

This step generates a memory dump of the Virtual Machine volatile data with the help of a programmed module. First, the state of the Virtual Machine is paused, then the module interacts with the hypervisor through the VirtualBox API to request a memory dump. By means of the VirtualBox debugger it is possible to acquire memory data in ELF format by default, next the module extracts the first section of the ELF file that is called LOAD where the memory dump data is placed, this is possible by searching this section and its correct offset in the file, also the size of the data. Once the above is done, a raw file is created, this new file contains the memory image that can be analyzed for the next module, also the state of the Virtual Machine is resumed.

5.3 Analysis

The goal of this step is to solve the problem of the semantic gap by using the Volatility Framework. It takes the raw file obtained in the previous step and interprets the low-level bytes into high-level information by obtaining the process list from memory dump, number of threads, network connections and open files of them. Also, this step determines and choose the correct profile associated with the operating system of the Virtual Machine to be monitored.

5.4 Logging

It generates a log in plain text that contains the name of the processes, their identifiers, identifiers of the parent processes, the number of threads, network connections and open files associated with them within the Virtual Machine through the monitoring time, this step is automated by means of a programmed module. Each log is created inside a directory that receives the name of the Virtual Machine monitored and each log is named with the start date and time of the monitoring.

5.5 Containment

The last step requires minimal human effort to determine that the information delivered by the generated logs, in each monitoring, report the presence of unknown processes by the security administrator, which can be malicious, once this has been detected, then he, through the implementation of this module, executes a command to kill the execution of the process inside the Virtual Machine. The COM/XPCOM interprocess mechanism is the responsible for sending the command form outside to the monitored Virtual Machine and guarantee its execution.

6 Experimental Results

In order to test the operation and functioning of the approach, three experiments were performed, each of them consisted in the execution of a different sample of malware, Table 1, inside the monitored Virtual Machine. The experiments were performed with physical machine of Intel ® core (TM) i7 CPU @ 2.60 GHzx4, 64-bit Kali GNU/Linux

Rolling as operating system with 8 GB in memory and a Virtual Machine with 32-bit Microsoft Windows XP SP3 as guest operating system with 1 GB in memory.

Table 1. Main characteristics of malware samples.

Experiment	Malware	MD5	Antivirus detection from VirusTotal
1	Trojan	7583a73f73638d23298ddb4900def643	56/64
2	Trojan	8915452ee0b8e754ee7b047a849a01a2	58/68
3	Trojan	c334b788e3da78c413364ef1e163b8ff	43/68

Using the method described in section 4, the approach started with the access module that selected the Virtual Machine to be monitored. The acquire module got the raw dump file to be analyzed by the next module. The analyze module detected the correct profile to be used with the Volatility Framework on the Virtual Machine, it was the “WinXPSP3x86” profile. It extracted relevant data of each sample of malware such as their identifiers, the number of their threads, network connections and open files.

The reporting module generated each monitoring log, the results obtained by this module, Table 2, allowed knowing the behavior of each sample of malware during its execution within the Virtual Machine.

In the first experiment the introduced malware executed 7 threads, 4 network connections and 24 open files. The four network connections correspond to IP addresses located in Canada, Italy, Pakistan and Germany, they are considered as part of blacklist according to VirusTotal. Among the total files opened by this malware, the suspicious files correspond to those whose functions are related to the Microsoft Windows

Table 2. Results reported by the reporting module.

Experiment	Malware	#Threads	Network connections		# Open files	Suspicious files
			Port	Address		
1	Trojan	7	8143	199.7.136.88	24	ws2_32.dll, ws2help.dll, mswsock.dll, dnsapi.dll, 996E.exe.
			1743	151.80.142.33		
			243	202.69.40.173		
			7447	78.47.66.169		
2	Trojan	5	1029	0.0.0.0	21	wsock32.dll, ws2_32.dll, crypt32.dll, autoexec.bat.
3	Trojan	1	80	211.104.175.45	12	ws2_32.dll, mswsock.dll, wshtcpip.dll, rpcrt4.dll, dnsapi.dll.

Sockets API, as well as to the file that contains these functions to establish connections to the Internet, a file was also found that is responsible for the resolution of domain names to their corresponding IP addresses. The file that is called 996E.exe, was opened by the malware, which is known to be used by Windows operating system to assure some specific programs to run properly, however the malicious code uses this file to circulate its own infectious files through its created threads and make the user believe that it is a benign software.

In the second experiment 5 threads, 1 network connection and 21 open files were registered, the only connection made by the malware corresponds to the IP address 0.0.0.0, the malware that uses this logical address do so in order to establish a connection remote from the attacking host, regardless of whether the victim host has multiple network interfaces, in this way, perform other actions such as establishing an ftp or ssh connection, related to it, the suspicious files opened by the malware allow to establish network connections and one of them contains encryption functions typical of the Microsoft Windows API. The file that is called autoexec.bat is altered by the malware to establish its self-execution with each start of the operating system.

The last experiment consisted in the execution of the third sample of malware, which was associated with the creation of 1 thread, 1 network connection and 12 open files, the IP that corresponds to its unique connection is located in South Korea. Its open files are related to network application activities and the port used to connect to the IP is port 80, typical of the HTTP protocol, it also uses the Remote Procedure Call API for communication with the Internet and the module that develops the resolution of domain names, translates the following URL registered in the log: http://download.everytoolbar.co.kr/setup/everytoolbar2_setup.exe to the aforementioned remote IP address, which is considered part of the blacklist stored in VirusTotal.

With the last module, of the last step, each malware sample execution, in each experiment, was killed from outside by means of this programmed module and as a consequence, each of the processes associated with them finalized their execution inside the Virtual Machine, this was achieved with the implemented “Kill” command concatenated with the process identifier which was reported in the log monitoring.

7 Conclusions and Future Work

In this paper, a method for malware analysis based on the Virtual Machine Introspection technique was proposed with the design of a 5 steps method: access, collection, analysis, logging and containment. By analyzing the memory image of the Virtual Machine, the behavior of three samples of malware in the Virtual Machine such as their identifiers, identifiers of their parent processes, their number of threads, also their network connections and open files of them were obtained. This information was reported in a log monitoring and it allowed to the security administrator the ability to kill a process, from outside the Virtual Machine, which he could consider as malicious. In this way, the experiments results verify the hypothesis that is possible to identify malware by means of Virtual Machine Introspection technique and solving the semantic gap problem by using the Volatility Framework to interpret the low-level bytes into high-level information

There are some aspects that should be improved as: Apply the method on the Linux guest operating system, find a better way to locate important structures in the volatile memory dump to replace the Volatility framework with one proposed and with the steps to analyze malware from memory dump, extend the method to identify rootkits in the Linux guest operating system.

Acknowledgment. The authors are thankful to the Instituto Politécnico Nacional (IPN) and Consejo Nacional de Ciencia y Tecnología (CONACyT) for their support in the development and achievement of this research.

References

1. López, C., Guadrón, R.: Computer forensics. In: 2016 IEEE 36th Central American and Panama, pp. 1–6. IEEE, San Jose, Costa Rica (2016)
2. Riaz, H., Ashraf, M.: Analysis of VMware virtual machine in forensics and anti-forensics paradigm. In: 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pp. 1–6, IEEE, Antalya, Turkey (2018)
3. Thongthua, A., Ngamsuriyaroj, S.: Assessment of Hypervisor Vulnerabilities. In: 2016 International Conference on Cloud Computing Research and Innovations (ICCCRI), pp. 71–77, IEEE, Singapore, Singapore (2016).
4. Huseinović, A., Ribić, S.: Virtual Machine Memory Forensics. In: 2013 21st Telecommunications Forum Telfor (TELFOR), pp. 940–942. IEEE, Belgrade, Serbia (2013)
5. Guangqi, L., Lianhai, W., Shuhui, Z., Shujiang, X., Lei, Z.: Memory Dump and Forensic Analysis Based on Virtual Machine. In: 2014 IEEE International Conference on Mechatronics and Automation, pp. 1773–1777, IEEE, Tianjin, China (2014)
6. Souppaya, M., Scarfone, K.: Guide to Malware Incident Prevention and Handling for Desktops and Laptops. NIST Special Publication 800-83 Rev 1 (2013)
7. Liu, J., Wang, Y., Wang, Y.: The Similarity Analysis of Malicious Software. In: 2016 IEEE First International Conference on Data Science in Cyberspace (DSC), pp. 161–168, IEEE, Changsha, China (2016)
8. Zhang, D., Zhang, Z., Jiang, B., Tse, T.: The Impact of Lightweight Disassembler on Malware Detection: An Empirical Study. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 620–629, IEEE, Tokyo, Japan (2018)
9. Kan, Z., Wang, H., Xu, G., Guo, Y., Chen, X.: Towards Light-Weight Deep Learning Based Malware Detection. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), pp. 600–609, IEEE, Tokyo, Japan (2018)
10. Li, N., Li, B., Li, J., Wo, T., Huai, J.: vMON: An Efficient Out-of-VM Process Monitor for Virtual Machines. In: 2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, pp. 1366–1373, IEEE, Zhangjiajie, China (2013)
11. Nemati, H., Dagenais, M.: VM processes state detection by hypervisor tracing. In: 2018 Annual IEEE International Systems Conference (SysCon), pp. 1–8. IEEE, Vancouver, BC, Canada (2018)
12. Oracle VM VirtualBox.: Programming Guide and Reference. Oracle Corporation (2018)
13. Volatility Foundation.

14. Jiang, S., Cai, H.: Monitoring VirtualBox Performance. In: Department of Computer Science and Engineering, University of Notre Dame, pp. 1–6, University of Notre Dame, Notre Dame, USA (2012)
15. Ajay, M., Jaidhar, C.: Virtual Machine Introspection based Spurious Process Detection in Virtualized Cloud Computing Environment. In: 2015 International Conference on Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), pp. 309–315, IEEE, Noida, India (2015)
16. Hebbal, Y., Laniepce, S., Menaud, J.: K-binID: Kernel binary code identification for Virtual Machine Introspection. In: 2017 IEEE Conference on Dependable and Secure Computing, pp. 107–114, IEEE, Taipei, Taiwan (2017)
17. Qiang, W., Xu, G., Dai, W., Zou, D., Jin, H.: CloudVMI: A Cloud-Oriented Writable Virtual Machine Introspection. In: IEEE Access, IEEE, vol. 5, pp. 21962–21976, National Natural Science Foundation, China (2017)
18. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: Network and Distributed Systems Security Symposium, pp. 191–206, NDSS, San Diego, California, USA (2003)
19. Hua, Q., Zhang, Y.: Detecting Malware and Rootkit via Memory Forensics. In: 2015 International Conference on Computer Science and Mechanical Automation (CSMA), pp. 92–96, IEEE, Hangzhou, China (2015)
20. Wei, C., We, J., Chieh, S., Kuo, S.: Memory forensics using virtual machine introspection for Malware analysis. In: 2017 IEEE Conference on Dependable and Secure Computing, pp. 518–519, IEEE, Taipei, Taiwan (2017)