A Method for Malware Detection in Virtualization Environment

Darshan Tank[1(✉)], Akshai Aggarwal[2], and Nirbhay Chaubey[3]

[1] Government Polytechnic (Rajkot), Gujarat Technological University, Ahmedabad, India
dmtank@gmail.com
[2] School of Computer Science, University of Windsor, Windsor, Canada
akshai.aggarwal@gmail.com
[3] Ganpat University, Gujarat, India
nirbhay@ieee.org

Abstract. Security of Virtual Machines (VMs) is a major concern with the virtualization environment. Virtual machines are a primary target for an adversary to acquire unethical access to the organization's virtual infrastructure. Traditional security measures are not enough for advanced malware detection. Today's advanced malware can easily avoid detection by using a number of evasion tactics. Process or code injection is one such technique to evade the detection of malware. Various process injection techniques are employed by malware to gain more stealth and to bypass security products. Detection of process injection attack is achieved with little effort on a physical machine as compared to a virtual machine. In this paper, we propose a novel approach to detect malware based on API function calls. API function call information has the ability to trace malware behavior. We found that the presence of certain API function calls may confirm the existence of malware. We implement dynamic malware analysis using the volatility framework.

Keywords: API function call · Hypervisor · Malware detection · Process injection · Virtual machine · Virtualization

1 Introduction

Cloud computing emerging as a future computing model. Virtualization is a key underlying technology to enable cloud computing. Virtualization creates and runs multiple VM or guest operating systems on a single physical machine using Virtual Machine Monitor (VMM) or Hypervisor. Hypervisor facilitates the abstract of physical machine resources such as CPU, Memory, I/O and NIC, etc., among several virtual machines. The sharing of resources increases the security challenges for the cloud service provider. The proliferation of unknown malware and sophisticated rootkits, are more prevalent to tamper the critical kernel data structures. Traditional In-host anti-malware solution is inadequate to ensure the security of the guest operating system, particularly in a virtualized environment [1]. VMI is able to gather the state information of the running VMs while functioning at VMM. Obtaining meaningful state information such as process list,

N. Chaubey et al. (Eds.): COMS2 2020, CCIS 1235, pp. 263–276, 2020.
https://doi.org/10.1007/978-981-15-6648-6_21

kernel driver module, etc., from the viewable raw bytes of the live guest virtual machine memory named as semantic gap [2].

One of the main reasons for using introspection in malware detection is that malware using advanced techniques such as rootkits are not detected using traditional automated malware-detection systems. The other reason is the advanced features that this technology provides, which allows the user to have a deep insight into each action happening in the virtual machine [3].

The rest of the paper is organized as follows: Sect. 2 provides background information on virtual machine introspection, memory mapping under a hypervisor and also introduces the concept of rootkits. Section 3 presents the related work of using the VMI technique to detect and characterize known and unknown malwares. Section 4 outlines LibVMI as a VM Introspection tool. Section 5 introduces volatility as a memory analysis framework. The architecture of our proposed malware detection method is described in Sect. 6. Evaluation and preliminary results are presented in Sect. 7. Finally, the conclusion and future work are discussed in Sect. 8.

2 Background

2.1 Virtual Machine Introspection

Virtual machine introspection is the process of observing the runtime state of virtual machines. Introspection can be achieved either from the hypervisor or from some virtual machine other than the one being supervised. VMI is an art of safeguarding a security-critical application running on virtual machines from security attacks [4]. VMI-based approaches are widely adopted for security applications, software debugging, and systems management. One can introspect the VM from inside or outside of the VM. VMI-based tools may be located inside or outside of the VM. VMI tools can also be used for malware analysis to analyze the behavior of the malware and to detect the latest malware attacks. VMI coupled with existing virtual infrastructure management solutions can become a powerful tool for memory analysis and event correlation. The semantic gap is one of the main restraints of virtual machine introspection [5]. In a virtualized environment, the semantic gap can be defined as the extraction of high-level information of guest OS state from low-level information obtained externally at the hypervisor level [6]. One can do introspection within the virtual machines or outside the virtual machines.

2.2 Memory Mapping

In a normal scenario, there are two levels of memory: virtual memory and physical memory of the physical machine. But when we talk about hypervisors, there are three levels of memory: virtual memory and physical memory of the virtual machine, and physical memory of the host machine. The hypervisors only allocate memory to the virtual machine. By default, hypervisors have no knowledge of what types of activities being performed inside the virtual memory of the virtual machine. To get that information, additional tools have to be installed. Below is a generalized example of memory-sharing within the virtual machine. Figure 1 shows the three levels of memory addressing under hypervisor [7].

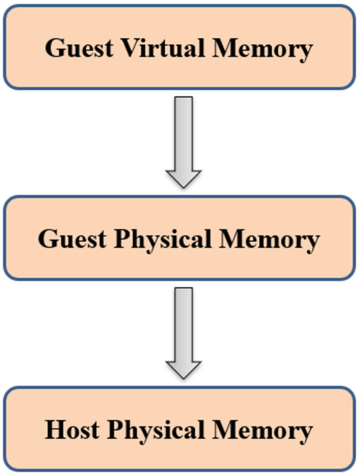

Fig. 1. Memory mapping under the hypervisor

One of the primary objectives of the VMI tools is to translate the memory addresses of the virtual machine's virtual memory: first, from the virtual to the physical memory of the virtual machine, then to the physical memory of the host machine. This will help the hypervisor to access the correct memory area during introspection.

2.3 Rootkits

Rootkits are malwares allowing permanent or consistent, undetectable presence in a computer system. Rootkits can hide specific system resources to achieve the goal of hiding the intrusion into the compromised computer. Rootkits deviate the normal behavior of the system by injecting malicious code into an operating system [8]. Kernel rootkits execute in privileged mode on Ring 0, making it very hard to detect. Kernel rootkits have posed serious security threats due to their stealthy manner. More advanced rootkits can launch Direct Kernel Object Manipulation (DKOM) attacks, which directly modify the core data structure of the OS kernel in memory. Malicious library injection and code injection are also common means for rootkits to subvert the system.

3 Related Work

Researchers and security experts have introduced many ideas and prototypes for malware detection and classification. Malware detection methods can be categorized into two classes: Signature-based static analysis and Behavirol-based dynamic analysis. Static analysis is accomplished without executing the samples while dynamic analysis is performed by executing samples in the virtualization environment. Huseinovic et al. [9] proposed a process monitoring mechanism in a VMware VM running WindowsXP. Hua et al. [10] have designed and implemented a process detection system called VmRecoverySystem. Their proposed architecture uses KVM as a hypervisor which consists of four

modules. Tien et al. [11] introduced a VMI method to monitor the presence of malware in the volatile memory of the VM through the analysis of its processes, files, registers, and network activities. Case et al. [12] presented a new kernel-based rootkit detection technique applicable to the Mac OS X system. They have used the most popular memory forensic framework Volatility to analyze the features of malwares.

For detecting malwares in Android, Yang et al. [13] proposed a general tool named AMExtractor for volatile memory acquisition for Android devices. For malware detection in a virtualization environment, Kumara et al. [14] leveraged memory forensic tools such as Volatility and Rekall to analyze the memory state of the VMs, which can address the semantic gap problem existing in VMI. Hua et al. [10] designed and implemented a VMM-based hidden process detection system to investigate rootkits by identifying the lack of the critical process and the target hidden process from the aspect of memory forensics. Tien et al. [15] introduced a memory data monitoring method against the running malware outside the VM, various features were observed from the memory. Kumara et al. [16] proposed an automated multi-level detection system for Rootkits and other malwares for VMs at the hypervisor level. Mosli et al. [17] proposed an automated malware detection method using artifacts in forensic memory images. Kumara et al. [18] proposed an advanced VMM-based machine learning technique at the hypervisor. Machine learning techniques were highly used to analyze the executables that were mined and extracted using MFA-based techniques. Tank et al. [19] presented a review of Mobile Cloud Computing (MCC), its security & privacy issues and vulnerabilities affecting cloud computing systems, analysed and compared various possible approaches proposed by the researchers to address security and privacy issues in MCC. Tank et al. [20] analyzed security issues in an open-source cloud computing project - OpenStack Keystone. Tank [21] identified a need for a lightweight secure framework that provides security with minimum communication and processing overhead on mobile devices. Tank et al. [22] presented a critical study and comparison of virtualization vulnerabilities, security issues, and solutions. Tank et al. [23] discussed Cache Side Channel (CSC) attacks as prominent security threats and introduced a novel approach to detect cache attacks in virtualized environments. Tank et al. [24] explored virtualization aspects of cybersecurity threats and solutions in the cloud computing environment. From the above researches, one can conclude that live memory analysis is an effective way to detect advance malwares.

4 LibVMI - VM Introspection Tool

LibVMI is an open source introspection library. LibVMI focuses on writing and reading memory from VMs. LibVMI is an extended version based on XenAccess Library. LibVMI is designed to work across multiple platforms [25]. LibVMI allows accessing the memory of running virtual machines. In addition to memory access, LibVMI also supports memory events. LibVMI can be utilized to bridge the semantic gap between the hypervisor and guest operating systems [26]. It offers the following features.

- Easily extensible and optimized performance.
- It provides near-native speeds.

- Address the semantic gap problem.
- Access a VM's state from outside of the VM and broad platform support.

5 The Volatility Framework

Volatility is an advanced memory analysis framework. It supports analysis for Linux, Windows, Mac, and Android systems [27]. Various volatility plugins are also developed and maintained by the community to extract information from memory samples. Volatility can be utilized as a memory forensic toolkit to detect advanced malware with a real case scenario [28]. The volatility framework offers the following features.

- An advanced & open source memory analysis tool.
- Support live analysis of virtual machines.
- Runs on Linux, Windows, Mac, and Android systems.
- It can be used to detect advanced malware with a real case scenario.
- Support a variety of file formats.
- Plugins can be developed and distributed independently.

The volatility tool supports a wealth of perceptions into the working of a system [29]. We used Volatility 2.6.1 in our research to extract higher-level semantic information from the live Windows 7 virtual machine. The LibVMI also adds improved integration with Volatility [30].

6 Proposed Malware Detection Method

Malware refers to malicious programs. In this work, we propose a method for malware detection based on examination of API function calls and API function calls sequences. We monitor API function calls and function calls sequences indicative of various types of process injection attacks. The extracted API function calls to be represented as a feature of the machine learning model. Various malware injectors are executed on Windows virtual machines and their runtime memory is acquired. Behavior-based dynamic analysis is carried out using a volatility framework.

Dynamic malware analysis is performed using the Volatility framework. We use impscan [31] and procdump [32] volatility commands. The impscan command is used to extract API function calls from the memory image. The procdump command is used to find the base address of the process. We make use of VirtualAllocEx and VirtualAlloc API functions as Indicators of Compromise (IoC) or malicious activity. The VirtualAllocEx and VirtualAlloc [33] functions allow to allocate memory in the address space of another process. We utilized VirtualAllocEx and VirtualAlloc functions as a precursor to code injection because malware needs to create space in the victim process. Figure 2 shows the generic architecture of our proposed malware detection approach.

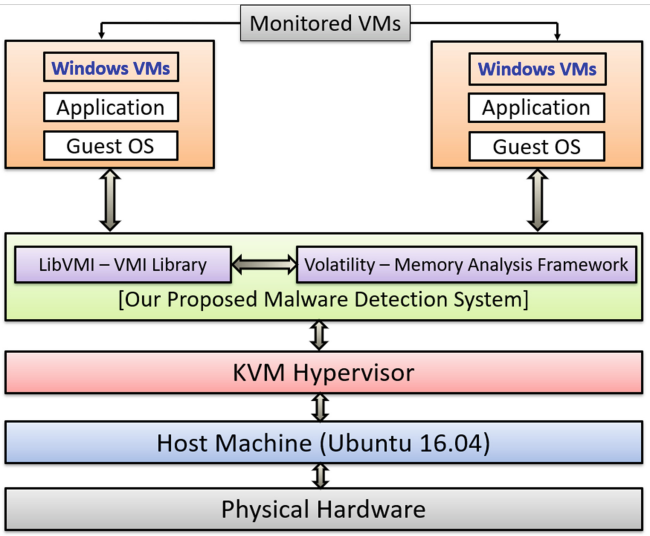

Fig. 2. The architecture of the proposed approach

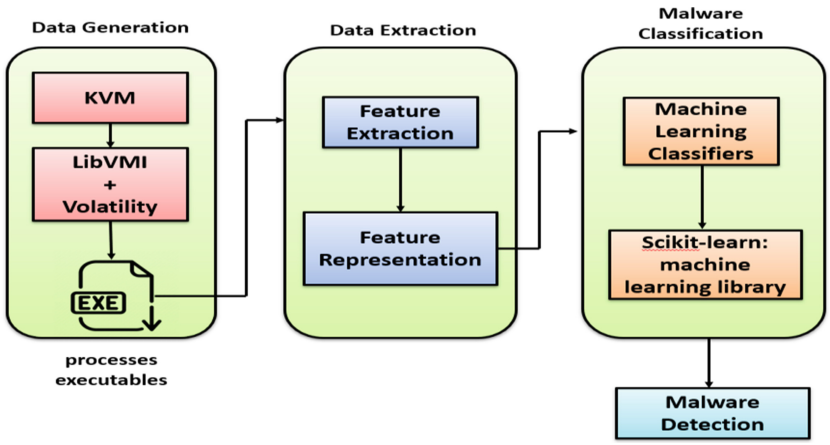

Fig. 3. Work flow of our proposed malware detection process

The work flow of our proposed malware detection process is depicted in Fig. 3. The malwares were executed on Windows virtual machines and their memory is acquired. Dynamic malware analysis is performed using the Volatility Framework. The impscan command from the volatility tool is used to extract API function calls from the dumped memory image. In the memory, the API function calls existed in the Import Address Table (IAT). The impscan command scans the memory image looking for API function

calls in the IAT table. The procdump command can be used to find the base address of the process.

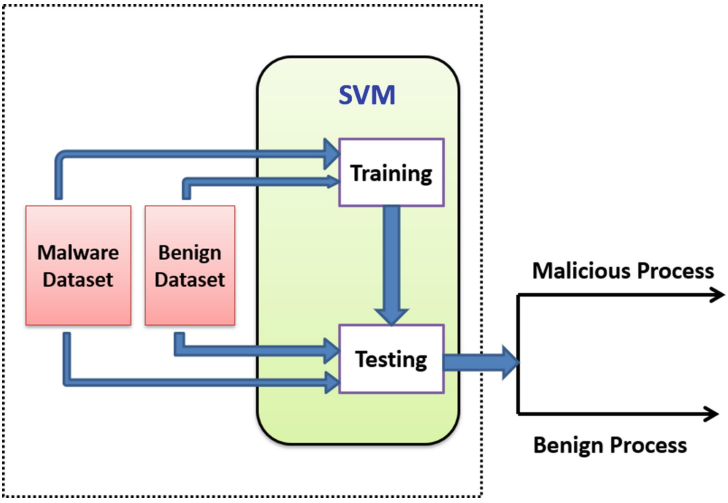

Fig. 4. Classification process using SVM binary classifier

The extracted Windows API function calls utilized as features of the machine learning model. We employed a machine learning method for the classification process. We used scikit-learn, a machine learning library in python. Scikit-learn features various clustering, regression, and classification algorithms including SVM, RF, GB, k-means, and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy [34]. We applied SVM (Support Vector Machine) supervised machine learning technique for the classification process. It classifies the given sample as either benign or malicious class as shown in Fig. 4.

7 Experimental Setup and Preliminary Results

We used the Kernel-based Virtual Machine (KVM) as our Virtual Machine Monitor (VMM). We perform experiments on the host system, which had the specifications shown in Table 1.

LibVMI python bindings (version-3.4) integrated with the Volatility framework (version-2.6.1) set up on the host operating system. Virtual Machines launched by the KVM hypervisor have Windows 7 and Ubuntu 12.04 guest OS running on it. We gathered experimental data from multiple scenarios. We divided the overall scenarios into two classes, a positive class which represent malware injector scenarios and a negative class which represents standard operations running on a virtual machine.

Table 1. Testbed configurations

Host OS	Ubuntu 16.04.6 LTS
Host OS Type	64-bit
Linux Kernel	Linux 4.15.0-74-generic
Architecture	X86_64
Processor	Intel(R) CoreTM i5-8265U CPU @ 1.60 GHz x 8
Disk	1 TB
Number of Cores & Threads	4 & 8
Physical Memory (RAM)	8 GB
Hypervisor (VMM)	KVM
VM – 1	OS – Windows 7, vCPU - 1 Memory – 2 GB, Storage – 30 GB
Tools /Framework Used	LibVMI python bindings (version-3.4) & Volatility framework (version-2.6.1) (Both are open source tools)

Table 2 highlights all collected scenarios for experimental evaluation. We run various process injection techniques collected from different Github repos to extract data for the positive class. In the idle condition, the virtual machine runs standard operations. We extracted the Windows API function calls from a dumped memory image and utilized it as features of the machine learning model.

Table 2. List of collected scenarios for evaluation

Positive class scenario	Negative class scenario
GitHub - theevilbit/injection [35]	Runs standard operations (Idle Condition)
GitHub - fdiskyou/injectAllTheThings [36]	
GitHub - secrary/InjectProc [37]	
GitHub - marcosd4h/memhunter [38]	
GitHub - stephenfewer/ReflectiveDLLInjection [39]	

The above figures show snapshots captured from live VM. Figure 5 shows a list of active VM on our host & acquiring a memory sample of the live VM. Figure 6 shows the working of the imageinfo command to identify the profile. Figure 7 shows working of procdump command to dump a process's executable and to get the base address of process. Figure 8 shows working of impscan command to extract API function calls from the memory image.

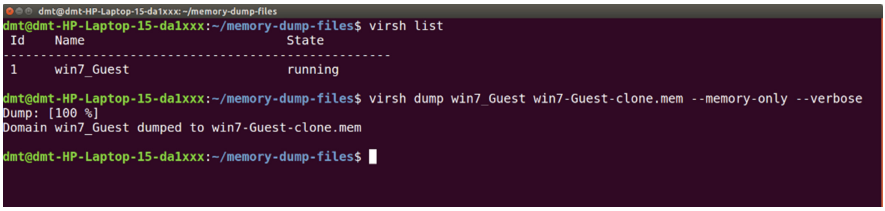

Fig. 5. List of active VM on our host & acquiring a memory sample of the live VM

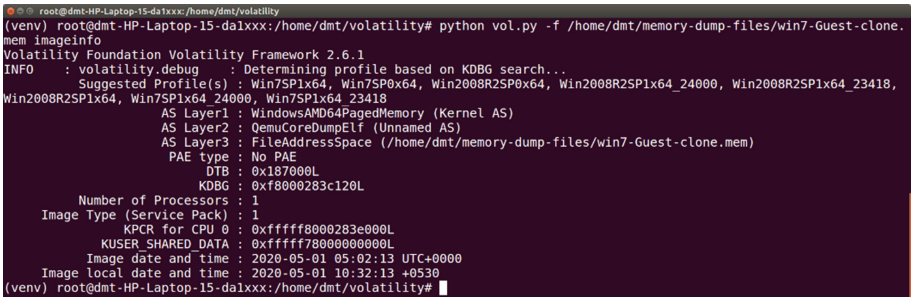

Fig. 6. imageinfo command used to identify the profile

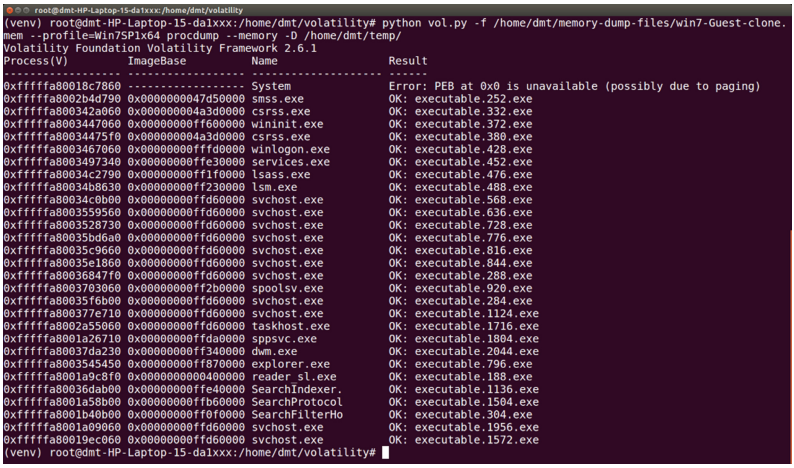

Fig. 7. procdump command to dump a process's executable & to get the base address of the process

```
root@dmt-HP-Laptop-15-da1xxx:/home/dmt/volatility

(venv) root@dmt-HP-Laptop-15-da1xxx:/home/dmt/volatility# python vol.py -f /home/dmt/memory-dump-files/win7-Guest-clone.
mem --profile=Win7SP1x64 impscan -p 796 -b 0x00000000ff871000
Volatility Foundation Volatility Framework 2.6.1
IAT              Call             Module                Function
---------------- ---------------- --------------------- --------
0x00000000ff929000 0x000007fefcef44c0 ADVAPI32.dll          RegCreateKeyW
0x00000000ff929008 0x000007fefcf04240 ADVAPI32.dll          RegCloseKey
0x00000000ff929010 0x000007fefcf04210 ADVAPI32.dll          RegOpenKeyExW
0x00000000ff929018 0x000007fefcef3bd0 ADVAPI32.dll          RegGetValueW
0x00000000ff929020 0x0000000076dcb0f0 ntdll.dll             EtwEventWrite
0x00000000ff929028 0x0000000076dcada0 ntdll.dll             EtwEventEnabled
0x00000000ff929030 0x0000000076e36cc0 ntdll.dll             EtwGetTraceLoggerHandle
0x00000000ff929038 0x0000000076e36c80 ntdll.dll             EtwGetTraceEnableLevel
0x00000000ff929040 0x0000000076e36c40 ntdll.dll             EtwGetTraceEnableFlags
0x00000000ff929048 0x0000000076d8dcd0 ntdll.dll             EtwRegisterTraceGuidsW
0x00000000ff929050 0x0000000076dc37a0 ntdll.dll             EtwUnregisterTraceGuids
0x00000000ff929058 0x000007fefcefe6f0 ADVAPI32.dll          RegQueryValueExW
0x00000000ff929060 0x000007fefcefd644 ADVAPI32.dll          GetLengthSid
0x00000000ff929068 0x000007fefcefd8a0 ADVAPI32.dll          GetTokenInformation
0x00000000ff929070 0x000007fefcefd8c0 ADVAPI32.dll          OpenProcessToken
0x00000000ff929078 0x000007fefcefd5f0 ADVAPI32.dll          RegCreateKeyExW
0x00000000ff929080 0x000007fefcef4490 ADVAPI32.dll          RegSetValueExW
0x00000000ff929088 0x0000000076d96da0 ntdll.dll             EtwEventRegister
0x00000000ff929090 0x000007fefcee8bc0 ADVAPI32.dll          RegDeleteKeyExW
0x00000000ff929098 0x0000000076dc37a0 ntdll.dll             EtwUnregisterTraceGuids
0x00000000ff9290a0 0x0000000076d8b9f0 ntdll.dll             EtwTraceMessage
0x00000000ff9290a8 0x000007fefcef55a0 ADVAPI32.dll          RegOpenKeyW
0x00000000ff9290b0 0x000007fefceeafd0 ADVAPI32.dll          RegDeleteValueW
0x00000000ff9290b8 0x000007fefcefe910 ADVAPI32.dll          RegQueryInfoKeyW
0x00000000ff9290c0 0x000007fefcf04000 ADVAPI32.dll          RegEnumValueW
0x00000000ff9290c8 0x000007fefcefe9a0 ADVAPI32.dll          LsaOpenPolicy
0x00000000ff9290d0 0x000007fefcef3b84 ADVAPI32.dll          GetSidSubAuthorityCount
0x00000000ff9290d8 0x000007fefceffd90 ADVAPI32.dll          LsaClose
0x00000000ff9290e0 0x000007fefcef3bac ADVAPI32.dll          IsValidSid
```

Fig. 8. impscan command to extract API function calls from the memory image

7.1 Experiment: DLL Injection Detection

Remote DLL (Dynamic Link Library) injection or Classic DLL injection is a form of process injection where the injected item is a DLL that is loaded within the context of the remote process. The program that performs the injection is called an injector. In this experiment, we detect the injector process running on a virtual machine (Figs. 9, 10 and 11).

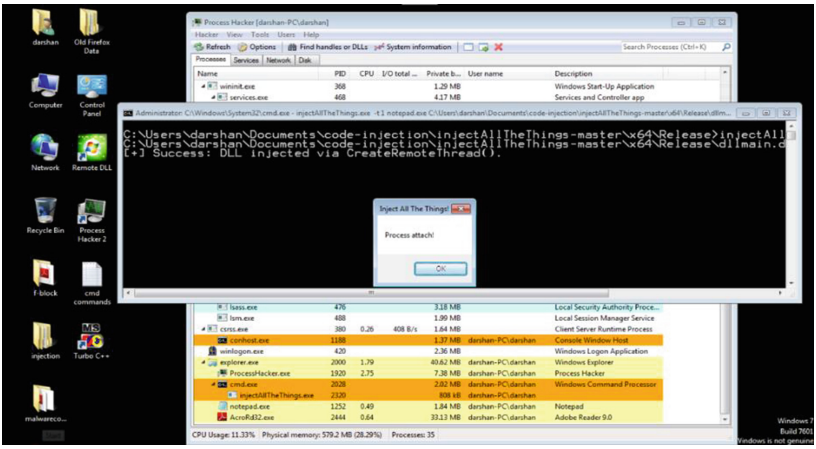

Fig. 9. The injector process is called through command prompt window

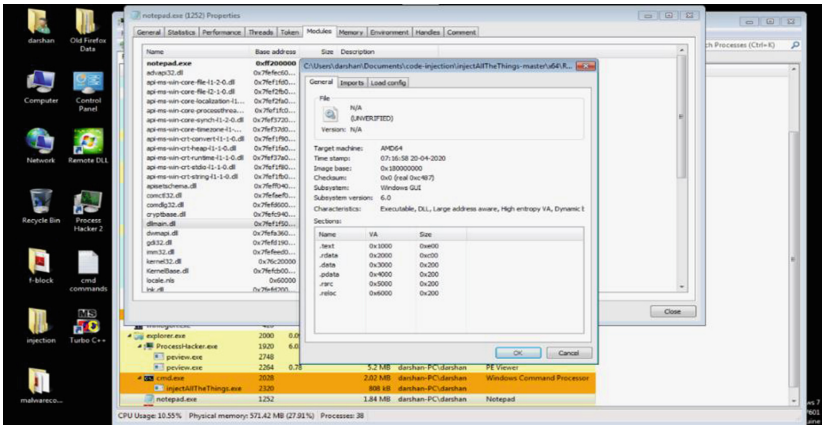

Fig. 10. Malicious DLL was loaded in process space

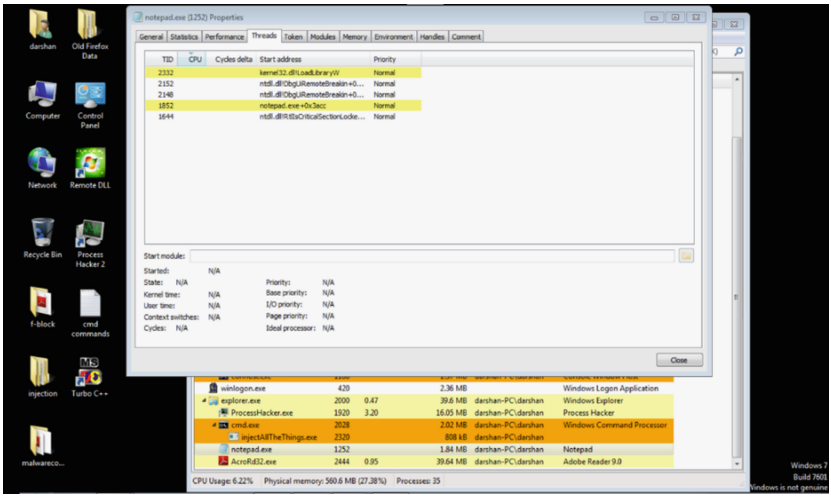

Fig. 11. The thread that loaded the DLL

We examine the captured memory image of VM to detect possible malicious DLL injection activities. The above images show how the injector process injects malicious DLL via the CreateRemoteThread function into the legitimate process. In the above images, the injector process is injectAllTheThings.exe, the injected process is notepad.exe and the injected DLL is dllmain.dll. As shown in the image the corresponding thread that loaded the DLL executes the LoadLibraryW API function.

We identified the malicious injector process by examining the process's API function call information from the captured memory image. Table 3 describes the identified malicious processes.

Table 3. Detection of malicious processes

Captured memory image	Process name	Process Id	Detected as
win7-Guest-clone.mem	injectAllTheThings.exe	2320	Malicious

8 Conclusion and Future Work

The malware leverages various process injection methods. Process injection attacks are the most damaging exploits faced by a large number of internet users today. Process injection or code injection techniques are used by malwares to gain more secrecy and to bypass employed security mechanisms by injecting malicious code that performs sensitive operations to a process that is privileged to do so. Detection of process injection attack is achieved with little effort on a physical machine as compared to a virtual machine. As there is no direct access mechanism to the physical memory of VMs in a virtualized environment, the detection of injector malwares running in user mode memory is more difficult.

In this paper, we introduce a new approach to detect malware running on virtual machine memory. Our objective of work is to detect malicious process injection activities running inside virtual machines based on API function call information. We successfully detected remote DLL injection using API function call details. As a containment plan, we can execute a command to kill the execution of malicious processes inside the VMs. We would like to automate the entire malware detection process. We also plan to measure the detection accuracy of our proposed method and to evaluate the robustness of our proposed system using publicly available known malware samples.

References

1. Ajay Kumara, M.A., Jaidhar, C.D.: VMI based automated real-time malware detector for virtualized cloud environment. In: Carlet, Claude, Hasan, M.Anwar, Saraswat, Vishal (eds.) SPACE 2016. LNCS, vol. 10076, pp. 281–300. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_16
2. Zhang, S., Meng, X., Wang, L., Xu, L., Han, X.: Secure virtualization environment based on advanced memory introspection. In: Security and Communication Networks (2018)
3. More, A., Tapaswi, S.: Virtual machine introspection: towards bridging the semantic gap. J. Cloud Comput. **3**(1), 16 (2014)
4. Rakotondravony, N., et al.: Classifying malware attacks in IaaS cloud environments. J. Cloud Comput. **6**(1), 26 (2017)
5. Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., Lee, W.: Virtuoso: narrowing the semantic gap in virtual machine introspection. In: 2011 IEEE Symposium on Security and Privacy, pp. 297–312. IEEE, May 2011
6. Fu, Y., Lin, Z.: Bridging the semantic gap in virtual machine introspection via online kernel data redirection. ACM Trans. Inf. Syst. Secur. **16**, 1–29 (2013). https://doi.org/10.1145/2516951.2505124
7. Virtual Machine Introspection in Malware Analysis. https://resources.infosecinstitute.com/virtual-machine-introspection-in-malware-analysis/. Accessed 17 Dec 2019

8. Wikipedia contributors: Rootkit. In Wikipedia, The Free Encyclopedia, 12 March 2020. https://en.wikipedia.org/w/index.php?title=Rootkit&oldid=945263481. Accessed 15 Mar 2020
9. Huseinovic, A., Ribic, S.: Virtual machine memory forensics. In: 2013 21st Telecommunications Forum Telfor (TELFOR), pp. 940–942 (2013)
10. 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 (2015)
11. Tien, C., Liao, J., Chang, S., Kuo, S.: Memory forensics using virtual machine introspection for Malware analysis. In: 2017 IEEE Conference on Dependable and Secure Computing, 518–519 (2017)
12. Case, A., Richard, I.I.I., Golden, G.: Advancing Mac OS X rootkit detection. Digital Invest. **14**, S25–S33 (2015). https://doi.org/10.1016/j.diin.2015.05.005
13. Yang, H., Zhuge, J., Liu, H., Liu, W.: A tool for volatile memory acquisition from android devices. DigitalForensics 2016. IAICT, vol. 484, pp. 365–378. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46279-0_19
14. Kumara, A., Jaidhar, C.D.: Execution time measurement of virtual machine volatile artifacts analyzers. In: 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), pp. 314–319. IEEE (2015)
15. Tien, C., Liao, J., Chang, S., Kuo, S.: Memory forensics using virtual machine introspection for Malware analysis. In: 2017 IEEE Conference on Dependable and Secure Computing, pp. 518–519 (2017)
16. Kumara, M.A., Jaidhar, C.D.: Automated multi-level malware detection system based on reconstructed semantic view of executables using machine learning techniques at VMM (2018)
17. Mosli, R., Li, R., Yuan, B., Pan, Y.: Automated malware detection using artifacts in forensic memory images. In: 2016 IEEE Symposium on Technologies for Homeland Security (HST), 1–6 (2016)
18. Kumara, M.A., Jaidhar, C.D.: Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. Digit. Invest. **23**, 99–123 (2017)
19. Chaubey, N.K., Tank, D.M.: Security, privacy and challenges in Mobile Cloud Computing (MCC): - a critical study and comparison. Int. J. Innov. Res. Comput. Commun. Eng. (IJIRCCE), **4**(2), 1259–1266 (2016). https://doi.org/10.15680/ijircce.2016.0402028
20. Tank, D., Aggarwal, A., Chaubey, N.: Security analysis of OpenStack keystone. Int. J. Latest Technol. Eng. Manag. Appl. Sci. (IJLTEMAS) **6**(6), 31–38 (2017)
21. Tank, D.M.: Security and privacy issues, solutions, and tools for MCC. In: Munir, K. (ed.) Security Management in Mobile Cloud Computing, pp. 121–147. IGI Global, Hershey (2017). https://doi.org/10.4018/978-1-5225-0602-7.ch006
22. Tank, D., Aggarwal, A. Chaubey, N.: Virtualization vulnerabilities, security issues, and solutions: a critical study and comparison. Int. J. Inf. Technol. (2019). https://doi.org/10.1007/s41870-019-00294-x
23. Tank, D., Aggarwal, A., Chaubey, N.: Cache attack detection in virtualized environments. J. Inf. Optim. Sci. **40**(5), 1109–1119 (2019). https://doi.org/10.1080/02522667.2019.1638001
24. Tank, D. M., Aggarwal, A., Chaubey, N.K.: Cyber security aspects of virtualization in cloud computing environments: analyzing virtualization-specific cyber security risks. In: Chaubey, N., Prajapati, B. (eds.), Quantum Cryptography and the Future of Cyber Security, pp. 283–299. IGI Global, Hershey (2020). https://doi.org/10.4018/978-1-7998-2253-0.ch013
25. Introduction to LibVMI. http://libvmi.com/docs/gcode-intro.html. Accessed 11 Jan 2020

26. Xiong, H. Liu, Z., Xu, W.: Libvmi: a library for bridging the semantic gap between guest OS and VMM. In: Proceedings - 2012 IEEE 12th International Conference on Computer and Information Technology, CIT 2012, pp. 549–556 (2012). https://doi.org/10.1109/cit.2012.119
27. An advanced memory forensics framework. http://volatilityfoundation.org/. Accessed 17 Nov 2019
28. Finding Advanced Malware Using Volatility. https://eforensicsmag.com/finding-advanced-malware-using-volatility/. Accessed 11 Jan 2020
29. Memory Forensics Investigation using Volatility. https://www.hackingarticles.in/memory-forensics-investigation-using-volatility-part-1/. Accessed 11 Jan 2020
30. Ainapure, B., Shah, D., Ananda Rao, A.: Performance analysis of virtual machine introspection tools in cloud environment. In: Proceedings of the International Conference on Informatics and Analytics (ICIA-16). Association for Computing Machinery, New York, NY, USA, Article 27, pp. 1–6 (2016). https://doi.org/10.1145/2980258.2980309
31. GitHub, volatilityfoundation/volatility - Command Reference Mal, https://github.com/volatilityfoundation/volatility/wiki/Command-Reference-Mal. Accessed 08 Jan 2020
32. GitHub, volatilityfoundation/volatility - Command Reference, https://github.com/volatilityfoundation/volatility/wiki/Command-Reference. Accessed 08 Jan 2020
33. VirtualAllocEx function (memoryapi.h) - Win32 apps | Microsoft Docs. https://docs.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualallocex. Accessed 08 Jan 2020
34. Wikipedia contributors. Scikit-learn. In Wikipedia, The Free Encyclopedia (2020). https://en.wikipedia.org/w/index.php?title=Scikit-learn&oldid=948478961. Accessed 11 Jan 2020
35. GitHub - theevilbit/injection. https://github.com/theevilbit/injection. Accessed 08 Jan 2020
36. GitHub - fdiskyou/injectAllTheThings: Seven different DLL injection techniques in one single project. https://github.com/fdiskyou/injectAllTheThings. Accessed 08 Jan 2020
37. GitHub - secrary/InjectProc: InjectProc - Process Injection Techniques. https://github.com/secrary/InjectProc. Accessed 08 Jan 2020
38. GitHub - marcosd4h/memhunter: Live hunting of code injection techniques. https://github.com/marcosd4h/memhunter. Accessed 08 Jan 2020
39. GitHub - stephenfewer/ReflectiveDLLInjection: Reflective DLL injection is a library injection technique in which the concept of reflective programming is employed to perform the loading of a library from memory into a host process. https://github.com/stephenfewer/ReflectiveDLLInjection. Accessed 08 Jan 2020