An Efficient Approach for Advanced Malware Analysis using Memory Forensic Technique

Mr. Chathuranga Rathnayaka
School of Computing, Engineering and Mathematics
Western Sydney University
Sydney, Australia
chathurangasa@yahoo.com

Dr. Aruna Jamdagni
School of Computing, Engineering and Mathematics
Western Sydney University
Sydney, Australia
a.jamdagni@westernsydney.edu.au

***Abstract*— Static analysis in malware analysis has been complex due to string searching methods. Forensic investigation of the physical memory or memory forensics provides a comprehensive analysis of malware, checking traces of malware in malware dumps that have been created while running in an operating system. In this study, we propose efficient and robust framework to analyse complex malwares by integrating both static analysis techniques and memory forensic techniques. The proposed framework has evaluated two hundred real malware samples and achieved a 90% detection rate. These results have been compared and verified with the results obtained from www.virustotal.com, which is online malware analysis tool. Additionally, we have identified the sources of many malware samples.**

Keywords—Malware, Static Analysis, Dynamic Analysis, Memory Forensic, Malware Detection

I. INTRODCUTION

Computer networks bring great changes to our lives. However, they have also provided a platform for malicious code propagation. Malwares have posed serious threats to the Internet security. In recent years, network security problems caused by malwares are increasing at an alarming rate and look terrible. To ensure network security, rapid change of technologies, operating systems, devices and files systems need to be closely monitored. In addition to that considerable amount of time is required to bring malwares to an understandable format.

Static and Dynamic malware analysis techniques are common in malware analysis. Malware analysis requires the knowledge of code or structure of the program to determine the processing functions. However, in the dynamic technique, dynamic or behavioral analysis is performed on malware that runs on the host system. In addition to that considerable tools are required for different levels of analysis.

Memory Forensics in malware analysis is becoming quite popular because it provides promising and reliable platform for advanced malware analysis. This approach is used by security analysts to investigate sophisticated malwares, root kits and cybercrimes. Memory Forensics in malware analysis is useful as it can be applied easily irrespective of technology, operating systems, devices and file systems. Usually, malware infections leave trails of evidence and symptoms in the memory. These symptoms help to identify artifacts related to the malware. Furthermore, they help in identifying suspected system processes, hidden or closed network connections, sockets, files, DLL modules, suspicious URL or IP addresses associated with the processes and suspicious strings associated with processes. They also help in extracting malicious processes by identifying attacker and their destination IP addresses, suspicious user profiles, registry entry, hooks to hide by themselves, malware injection to running processes, interdependencies of processes and the intended purpose of the attacks. The information in Random Access Memory (RAM) is cleaned once the computer is powered down. As a result, potential information regarding the malwares is lost, making it a challenging task to investigate the types of malware attack. Forensic Memory analysis plays a vital role in the analysis of malware because malware authors hide their foot print on victim's hard drive by executing and operating malwares in RAM. Volatility [9] which is a memory analysis tool is used to identify the nature of the malware attack and it provides successful entry point. The proposed work examines volatility of information in RAM which reveals flow of the malware, their existence and timeline in memory when they are executed.

In this study, we proposed modification of *"Enhancing Automated Malware Analysis Machines with Memory Analysis*" [4] approach. In Memory Forensics, it is not necessary to perform static analysis on malware samples, however in this study to reduce the encryption, obfuscated or packed nature in malware samples, static malware analysis is also performed on malware samples.

The remainder of this paper is organized as follows: Section II briefly reviews the literature related to this work. In Section III, we give the overview of the proposed framework and methodology of our proposed approach, which is applied for the analysis of advanced malware. Results are given in Section IV. Section V concludes our research work and suggests future research work.

2324-9013/17 $31.00 © 2017 IEEE
DOI 10.1109/Trustcom/BigDataSE/ICESS.2017.365

II. RELATED WORKS

The Enhancing Automated Malware Analysis Machines with Memory Analysis in [4] is the foundation for the work and mainly focused on memory analysis for advanced malware. In this approach malware analysis is performed in three stages. These stages are, classic memory analysis, interval-based memory analysis, and trigger-based memory analysis. The classical memory analysis is performed by running the malware in a controlled environment for 3 minutes. At the end of the execution, relevant dump files are collected for analysis. According to classic memory analysis they are unable to recognize the important information, because the malware clear memory region before it terminates and leaves no traces of the malware. In interval base method, six memory dumps are collected in 30 seconds time intervals for 3 minutes. This method has also do not leave any traces with regard to activities of the malware. In trigger-based memory analysis, controlled environment is configured for API-Based, Performance-Based and Instrumentation-Based [4] execution levels. API-based trigger approach has shown better results. The other important finding in this study is that they generate a clean non-malware infected, memory dump file. They use this dump file to compare with malware-infected dump files generated at different execution levels.

The other important and relevant information found from e-forensic blog in "*Finding Advanced Malware using Volatility*" [3], provides step by step approach to investigate critical and advanced malware. The work is divided in to two parts, memory acquisition and memory analysis. Overall work shows the importance of process Ids, mutex, file handless, hidden DLLs, virustotal submission, Kernel Callbacks and kernel API hooks using Volatility tool. It is a good handbook for incident response management too. These methods are very efficient and helpful in identifying several different types of malwares and their characteristics.

In our proposed research work, we have integrated both static analysis and memory forensic approaches and proposes an improvement of existing method [4], which provides better and reliable results. The reason for this is the resistance of some malware samples to reveal their intended behavior due to encryption and packed nature. Hence, both static malware analysis approach and memory forensic approaches need to work together to overcome behavioral and encryption issues. Volatility tool is the key tool and is used in our research work.

III. FRAMEWORK AND METHODOLGY

The overview of our proposed framework used for malware analysis is given in this section, where the framework and the malware sample analysis mechanism are discussed.

The whole malware analysis process consists of two major sections. Section A and section B respectively, as shown in Fig. 1. Section A is described in four steps (step 1, 2, 3, and 4). In Section B of Fig. 1, the incoming malware samples are uploaded to www.virustotal.com web interface which is a virus scan program integrated with several different antivirus engines in to a single web interface [15] for easy analysis. Finally, section A, result is verified with section B, result.

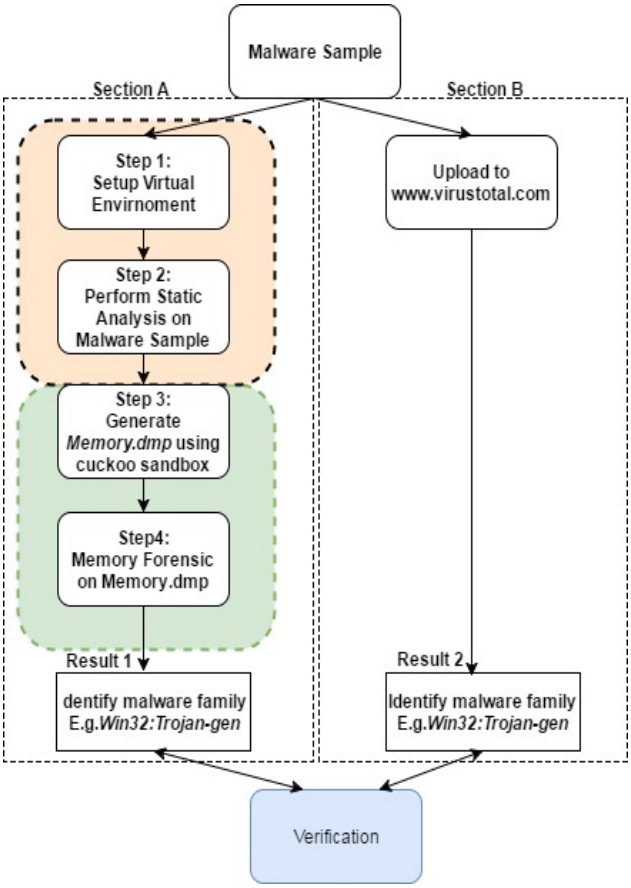

Fig. 1. The framework for malware family identification

In **Section A,** the incoming malware samples go through steps 1 to step 4. Static analysis of malware samples is performed in Step 1 and step 2, and forensic memory analysis is done in step 3 and step 4. The detail description of steps 1to step 4 is explained in the following sections.

Section A, **Step 1** is to create the virtual environment. In this study controlled environment is created using VMware. First install Ubuntu in VMware virtual machine

and make the host ready. Then create and install Virtual Box in Ubuntu. Install Windows XP in created Virtual Box. After installing Windows XP in Virtual Box, stop automatic update, Windows Firewalls and Virus guards to run the malware smoothly. Then install Cuckoo sandbox in Ubuntu according to the Cucuko installation guide [1]. Once cuckoo is successfully installed, install Volatility tools and necessary packages in Ubuntu. Now all installation parts are done. Then assign IP-addresses, add plugging, create Snapshots based on the scenario expected to be performed.

In **Section A, Step 2** static analysis is performed on the malware sample. In order to perform smooth execution, malware is reverse engineered. To full fill the requirement some static analysis tools are used to identify terminations in de-assembling the malware. Before each and every dump generation this task needs to be performed. In some instances due to hooks and functions with root-kits, it has been a tedious task to determine or bypass anti-debugger API functions. The following API calls **"ISDebuggerPresent", CheckRemoteDebuggerPresent", NtQueryInformationProcess"**, **"OutputDebugString"** are possible indications and first entry point to the malware.

According to the Fig. 2 the malware is checked for Process Environment Block (PEB) called BeingDebugged flag. The present of the "**BeingDebugging**" flag is a positive sign the malware is been debugged. In addition to that status is checked in "**ProcessHeap**" and "**NTGlobalFlag**" to determine the malware can run in a debugger. If they are debugged, bypass those using anti-debugging techniques. Now the malware is ready to go for memory forensic analysis. According to Section A, Step 4, malware is processed to identify malware family.

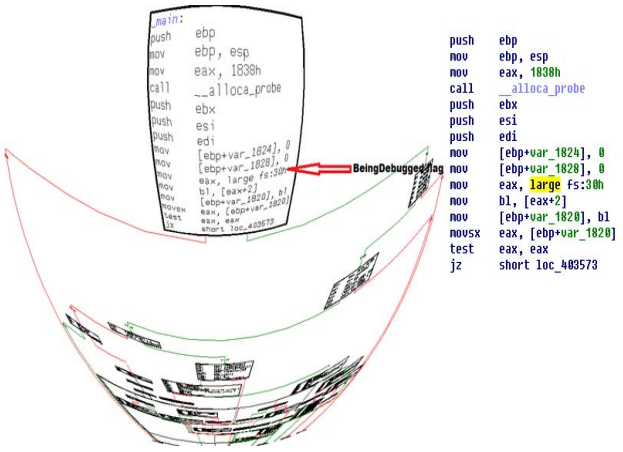

Fig. 2. Win32 graph view of the malware using IDA

Section A, Steps 3 and 4, are the final stages of the malware study. Here, Cuckoo Sandbox [8] which is a malware analysis system is used to obtain detail results in malwares. Finally malware execution takes place in Cuckoo Sandbox in order to generate memory dump. Memory forensic analysis is carried out on these dumps generated in step 3 for further identification and classification.

Finally, outcome of section A is compared with the outcome of section B to verify whether two outcomes are similar or not similar.

IV. Results and Analysis of Results

A. *Platform for Analysis*

VMware [5], Virtual Box [6], Ubuntu [7], Cuckoo Sandbox [8], Volatility tool [9], IDA pro [10], Wireshark [11], Virus Total [12] are the required tool for analysis.

We used two hundred malware samples for the testing of proposed framework. These malware samples downloaded from a popular malware website [13]. Most of the samples are packed and obfuscated. The dump generation process, described in Section A, Step 4, is shown in figure 1 is used for each and every malware variant. Our proposed method could identify 90% of malware samples successfully.

The malware sample used for explanation is: **"*Win32:Trojan-gen* (MD5: 14d4bdcc5f0fe917abde4b01e39a350e)".**

Based on our proposed method we obtained following information of the malware. They are Process information, log/ histories, PE file extraction, Networking Information, Kernal Memory, Kernal Objects, Processes Listings, Injected Codes, Registry, API Hooks, Strings, Timelines, Passwords, Files systems, Malware Specifications, User activities and Encryption. As such, investigation takes place in finding that information. The actual work contains more results, however the paper shows some crucial snaps resulted by the *memory.dmp* file. And results shows, work carried out related to a single malware sample as such results from two hundred different samples are taken into account to calculate the ratio.

Malware Name	Malware Type	Verified (Y/N)
Android.Spy.49_iBanking_Feb2014	Trojan.Android.Agent.cudjnc	Y
AndroRat_6Dec2013	Android.Riskware.AndroRat.	Y
Backdoor.MSIL.Tyupkin	Backdoor.W32.Tyupkin.d!c	Y
BlackEnergy2.1	TR/Crypt.EPACK.Gen2	Y
CryptoLocker_10Sep2013	Troj.Ransom.W32.Blocker.cfwh!c	Y
CryptoLocker_20Nov2013	Trojan[Ransom]/Win32.Blocker	Y
Dino	Trojan.Gen.6	Y
Dropper.Taleret	W32/GOSME.ANI!tr.bdr	Y
Duqu2	Trojan.Win64.Duqu2.dtaxgj	Y
EquationGroup	Trojan[Backdoor]/Win32.Laserv	Y
EquationGroup.DoubleFantasy	Win32.Trojan.WisdomEyes.16070401.9500.9942	Y
EquationGroup.EquationDrug	Dropped:Trojan.Generic.8262217	Y
EquationGroup.EquationLaser	Backdoor.Win32.Laserv.b	Y
EquationGroup.GrayFish	W32/FakeAlert.LP.gen!Eldorado	Y
EquationGroup.GROK	a variant of	Y
EquationGroup.TripleFantasy	malicious (high confidence)	Y
IllusionBot_May2007	Backdoor.Ircbot.XF	Y
Kelihos	HEUR:Trojan.Win32.Generic	Y
KRBanker	Trojan.Win32.TrjGen.bbmcln	Y
Linux.Chapros.A	Trojan.Apmod.bfnfeo	Y
Linux.Wirenet	Suspicious_GEN.F47V0323	Y
Neurevt.1.7.0.1	Trojan.Win32.Pincav.cqsImo	Y
Nivdort	HEUR:Trojan.Win32.Generic	Y
njRAT-v0.6.4	Trojan.Win32.DownLoader10.cvpcyl	Y
OSX.Wirenet	MAC.OSX.Backdoor.Wirenet.	Y
PlugX	Trojan.Win32.Gulpix.cujjsa	Y
Poweliks	Suspicious_GEN.F47V0321	Y
Ransomware.Cryptowall	Trojan.Win32.Panda.eahzta	Y
Ransomware.Jigsaw	Ransom.FileCryptor!8.1A7	Y
Ransomware.Locky	Trojan.Encoder.3976	Y
Ransomware.Petya	Trojan-Ransom.Win32.Petr.l	Y
Ransomware.Radamant	Malware.FakePDF@CV!1.A2	Y
Careto_Feb2014	Trojan/Win32.Careto.R9738	Y
Nitlove	Spyware.Infostealer.nitlovep	Y
EquationGroup.Fanny	Trojan.Win32.EquationDrug.	N
Keylogger.Ardamax	Dropped:Application.Keylogger.Ardamax.Gen (B)	N
Ransomware.Matsnu	Trojan-Ransom.Win32.Foreign!O	N
PotaoExpress	Trojan.Win32.Potao.dgbss	N
Dyre	Trojan.Win32.Dyre.efhazr	N
InstallBC201401.exe	Virus.Win32.Gen-Crypt.ccnc	N

Table 1. Tested Malware Sample List (40 samples shown out of 200)

According to table 1, forty malware samples out of two hundred is shown. Among them, 6 samples could not detect using existing tools. There are several reasons them to be undetected. Some malwares have the capability to detect sandboxing capability so they detect the running environment and stop their activities. As a result they remain as safe files. Some malwares generates encrypted payload which is hard to detect using existing tools and techniques. Polymorphic behavior (changing file name and compression) is another issue in detecting the malware. However, according to the figure 8, some of them were able to identify correctly and some went undetected. Moreover, some malware terminates in static analysis process. Most of them can be prepare for analysis using existing analysis techniques and very few of them still remain undetected.

B. Results obtained after Processing of sample in section B

We uploaded the sample to www.virustotal.com web interface for analysis and classification. Fig 3 shows the outcome of the sample. The snapshot indicates that the sample is win32mal.exe, this is a malware sample. The detection ratio (29/61) is calculated by www.virustotal.com scan engine. This engine classifies this sample as a ***Win32:Trojan-gen*** malware.

Fig. 3. Snapshot - Virus total detection ratio for win32mal.exe

C. Results obtained after Processing the sample in section A

Static analysis of malware sample is carried out in steps 2 and then memory dump is generated in step 3 for this sample. To identify the family of the malware sample following parameters are checked in step 4.

a. Malware Source Identification

According to Wireshark TCP stream shown in Fig. 4, identifies the entry point and shows how attack delivers by itself. Cuckoo *dump.pcap* file provide the source host from where malware has come. It is observed that it comes from www.abcd.com (changed the actual name). The current result does not show any public IP address manipulated during the execution however other malware consist of network operation and generate traffic to public IP address.

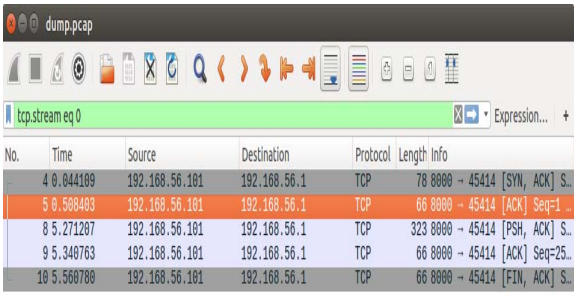

Fig. 4. Wireshark TCP stream

Note: All following results (snapshots, 5-9) are obtained using memory forensics in Section A, Step 4

b. Malware Resident in the System identification

Fig. 5 shows the result of duplicate files created by malware. The duplicate malware file, ***435be1c6e904836a_ win32mal.exe*** is generated by deleting the original malware file ***win32mal.exe.*** Further analysis is based on "*memory.dmp"*. Now ***win32mal.exe*** does not exist instead of that *435be1c6e904836a_ win32mal.exe* exists.

Fig. 5. Alternative file created from win32mal.exe

c. Malware residence in registry

The malware creates a registry value in the current location before it starts. Fig. 6 shows the registry path and the value created. This is also a good investigation to identify registry operation

Fig. 6. Registry value created

d. Malware Resident in the System identification

- *Identify possible DLL injections*

Fig. 7. Imports in KERNELL32.dll

According to the fig. 7, the trick of the malware in execution is observed. Check here if malware is associated with any type of DLL injections. DLL injection is common types of attack in modern malware. It can be identified based on ***"VirtualAllocEx", "CreateRemoteThread", "SetWindowsHookEx"*** API calls. These API calls need to be checked under imported API.

- *Suspected mutex identification*

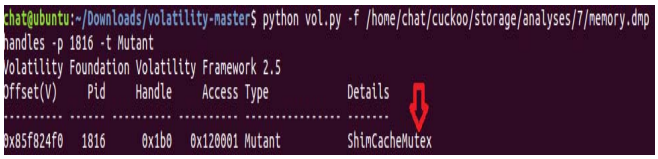

Fig. 8. Mutex created by malware win32mal.exe

Malware creates suspected "*mutex" as* in shown in Fig. 8 at the process in PID 1816.

e. Self-replication Identification

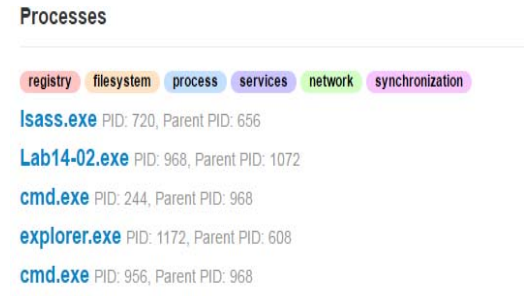

Fig. 9. Backdoor process created

Memory forensic in step 4 then identify the self-replication capabilities of the malware. Malware creates certain processes during the execution. These are backdoor processes running and have been replicated from the *win32mal.exe.* Fig. 9 describes the snapshot of backdoor process.

Further analysis takes place to identify how malware obtains certain level of privilege, affected file systems and registry values altered, documents or files system affected, how it resides in the network.

Finally based on suspicious DLLs obtained in step 4, malware family is identified as ***Win32:Trojan-gen.*** We verified the result achieved in section A with the result achieved in section B. The two results are same. Over the experiment of two hundred different malware samples, 90% of malware samples are identified correctly. We could identify 80% malware's target attack. We could successfully identify the locations of 20% of malwares from where they came. 10% samples were none-classified.

V. Conclusion and Future work

In this paper we presented malware analysis that integrates the static analysis of malware with forensic analysis of memory dumps. The advantage of this approach is that malwares, that resist in revealing their behaviors and do not show artifacts, can be analysed. We performed static analysis on malware samples to reveal their intended behavior, which were hidden, due to encryption and packed nature. We have observed that successful malware infections leave a trail of foot prints in the memory. This provides valuable entry points for the analysis of advanced malwares. Within the scope, we were able to achieve a 90% classification rate. This rate can be maximized based on different approaches in future experiments. Human errors during this experiment could minimize using best practices in a solid lab environment.

However, compared to static analysis, this method provides more features and artifacts for analysis because the dump is obtained after the real execution of the malware. Cuckoo sandbox provides integration with other tools such as Volatility, Yara, Virustotal, and Wireshark and provides an opportunity for better testing. The current approach has certain limitations in setting up the environment and in dumps generation, because the size of "*memory.dmp"* may exceeds the limit of 1GB for some instances.

For future work proposals, we will expand the same analysis by integrating Malware Attribute Enumeration and Characterization (MAEC) [14] techniques. This will allow the investigation and sharing of structural information for malware types based on attributes, and develops a common method which can be applied to new and advanced malwares.

References

[1] Cuckoo Sandbox Book. (2016) (2nd ed.). Retrieved from https://media.readthedocs.org/pdf/cuckoo/latest/cuckoo.pdf

[2] *E-Discovery & Computer Forensics*. (2016). *AccessData*. Retrieved 5 March 2017, from http://www.accessdata.com

[3] Duc, H. (2016). *Finding Advanced Malware Using Volatility - eForensics*. *eForensics*. Retrieved 5 March 2017, from https://eforensicsmag.com/finding-advanced-malware-using-volatility/

[4] Teller, T., & Hayon, A. (2014). Enhancing automated malware analysis machines with memory analysis. *USA: Black Hat*.

[5] VMware Virtualization for Desktop & Server, Application, Public & Hybrid Clouds. (2017). Vmware.com. Retrieved, from http://www.vmware.com/au.html

[6]Oracle VM VirtualBox. (2017). Virtualbox.org. Retrieved 2 March 2017, from https://www.virtualbox.org/

[7] The leading operating system for PCs, tablets, phones, IoT devices, servers and the cloud | Ubuntu. (2017). Ubuntu.com. Retrieved 2 March 2017, from https://www.ubuntu.com/

[8] Automated Malware Analysis - Cuckoo Sandbox. (2017). Cuckoosandbox.org. Retrieved 27 February 2017, from https://cuckoosandbox.org/

[9] The Volatility Foundation - Open Source Memory Forensics. (2017). The Volatility Foundation - Open Source Memory Forensics. Retrieved 2 March 2017, from http://www.volatilityfoundation.org/

[10] *Hex-rays*. (2017). *Hex-rays.com*. Retrieved 5 March 2017, from https://www.hex-rays.com/

[11] Wireshark · Go Deep. (2017). Wireshark.org. Retrieved 2 March 2017, from https://www.wireshark.org/

[12] VirusTotal - Free Online Virus, Malware and URL Scanner. (2017). Virustotal.com. Retrieved 2 March 2017, from http://www.virustotal.com

[13]M. Sikorski and A. Honig, Practical malware analysis. San Francisco: No Starch Press, 2012.

[14] Beck, D., Kirllov, I., & Chase, H P. (2014). The MAEC™ Language Overview. MITRE Corporation.

[15] *VirusTotal.(2016).En.Wikipedia.org,* Retrieved 18th March 2017, froToatl.m https://en.wikipedia.org/wiki/VirusTotal.