A Comparison of Windows Physical Memory Acquisition Tools

Waqas Ahmed, Baber Aslam
National University of Sciences and Technology, H-12, Islamabad, Pakistan
waqasahmed.msis11@students.mcs.edu.pk, baber-mcs@nust.edu.pk

Abstract—**Memory forensics analysis is an important area of digital forensics especially in incident response, malware analysis and behavior analysis (of application and system software) in physical memory. Traditional digital forensics, such as investigating non-volatile storage, cannot be used to establish the current state of the system (including network connections) or for analysis of malwares that use evasion techniques like encryption. Accurate activities of a program can only be analyzed when it is loaded in memory for execution, for which volatile memory forensics analysis is used. The success of memory forensics depends on the accuracy and completeness of memory image, which means all sections of memory (both kernel and user space) must be captured accurately. Several tools with varied capabilities and accuracies are available for capturing the memory. In order to determine the capabilities and accuracy of Windows volatile memory capturing tools, we have analyzed several different Windows volatile memory acquisition tools and have also compared their results. For analysis of captured memory, we used three different memory analysis tools. The resulting comparisons can be used by investigators to select tools as per their need.**

Index Terms—**memory forensics analysis; acquisition tools; anti-debugging; Karos online; pslist; psscan**

I. INTRODUCTION

Memory forensics is an area of digital forensics that deals with the examination of information stored in a system's physical memory. Every function/task performed by CPU can be found in physical memory such as user credentials, network connections, running processes, services, decrypted program, logged users information, running programs (including malwares), documents, registry keys, IP addresses, and web browser data [1]. This variety of information is lost due to volatile nature of physical memory or when the system is turnoff. Therefore analyzing Windows physical memory is useful for both criminal investigation and incident response due to importance of available evidences in volatile memory.

Memory is divided in two areas/modes: user area and kernel area. Operating system (OS) resides in kernel area whereas the user programs reside in user area. According to OS structure, a user program cannot directly access kernel memory. Therefore, memory forensics tools need special privileges to capture/dump kernel memory. Further, there are number of applications and malicious programs that use anti-debugging mechanisms, which prevent tools from dumping volatile memory. In addition, many applications also prevent their memory sets from dumping. These applications include games, malwares etc. Memory acquisition tools usually acquire empty memory set or garbage data when they try to dump volatile memory when an anti-debugging or anti-dumping system is running memory. Above discussion demonstrates that accurate and reliable memory acquisition plays vital role in memory and malware analysis [2].

In this paper we have analyzed six different volatile memory acquisition tools and compared their dumping accuracy. We used two scenario for comparison of these tools. The paper is organized in six sections. In section–II memory acquisition methodology is discussed, section-III gives details of experimental setup and memory acquisition scenarios, section-IV discusses tools examination, section V discuss the results and section VI gives conclusion.

II. MEMORY ACQUISITION METHODOLOGY

Memory acquisition or act of dumping physical memory is first phase in digital investigation. There are two approaches of acquiring memory namely hardware based acquisition and software base acquisition. Memory contents can be acquired though hardware like *Tribble* (via PCI Express card), *Firewire* (Direct Memory Access concept), *VMware* memory snapshot (virtualization environment), hibernation & page file, Windows crash dump and other software based acquisition. The details of hardware based acquisition and software based acquisition tools are discussed below:-

A. Hardware Based Acquisition

The "*Tribble*" was introduced in 2004 by Joe Grand and Brian Carrier Studio, Inc. It is a PCI expansion card to be installed on a machine prior to the incident or before acquisition needs to be planned [3].

Another hardware approach to acquire content of physical memory is "*Firewire*" attack on RAM. The *Firewire* uses *DMA* (Direct Memory Access) concept. In this method CPU is bypassed and contents of physical memory can directly be accessed through *Firewire* port. This technique allows high speed data transfer without any compatibility issue of different Operating System versions [4].

B. Software Based Acquisition

Hardware based acquisition has its limitations such as cost, prior installation on target system etc. These reasons make it difficult to use hardware based acquisition. Software based solutions are low cost, user-friendly and easily deployable. However, some software based solutions also have flaws, as already discussed in section-I.

Some popular software tools to acquire Windows physical memory include: *DumpIt* by *MoonSols*, RAM capture by *BelkaSoft*, *Memoryze* by *Mandiant*, *Access Data FTK Imager*, *Magnet RAM Capture* by *Magnet Forensics*, *Winpmem*, *OS Forensics*, *Redline* by *Mandiant*, *EnCase/WinEn*, *HBGary FastDump*, *F-Response*, *KnTTools* by *GMG Systems Inc*,

978-1-5090-0073-9/15/$31.00 ©2015 IEEE

KnTDD, *Mdd* or *Memory dd* by *ManTech*, etc. Scope of this paper is limited to first six tools only. We have used these tools to acquire the physical memory of Windows 7 systems. For analysis of the captured memory from these tools, we used three different memory analysis tools.

III. Experimental Setup

In this section we discuss the experimental environment and general procedure of memory acquisition and analysis. We have made two scenarios for memory acquisition from target, these scenarios are categorized as Scenario-I and Scenario –II.

A. Scenario-I (no anti-debugging tool installed)

In this scenario, we installed a fresh copy of *Windows 7 Ultimate 32 Bit Edition Service Pack 0* on laptop computer. After installing operating system and some application software, we restarted the system and when Windows booted/started properly, we captured volatile memory with a memory capturing tool. We used six different memory acquisition/capturing tools (as mentioned in Section-II) one by one in same scenario i.e., we restarted the system and captured its volatile memory after Windows have booted/started properly.

B. Scenario-II (anti-debugging tool installed)

In this scenario, we installed anti-debugging enabled game named *Karos online*, installed through *Steam* software on same *Windows 7 Ultimate 32 Bit Edition Service Pack 0*. *Karos online* game uses *nProtect GameGuard* for the purpose of anti-debugging and anti-cheating. *nProtect GameGuard* can be installed as an isolated software or built within other programs, as in case of many multiplayer online roleplaying games, to block common cheating techniques. *nProtect GameGuard* works similar to a rootkit, it hides the game application process and monitors complete memory range [5] and work as anti-debugging system, as a result of which, not every acquisition tool is capable of acquiring contents of physical memory completely or accurately. That's why we install *nProtect GameGuard* based game so that we can compare volatile memory acquisition tools accuracy.

After installing *Karos online* game we started playing game, and when the game was running, we captured volatile memory with six memory acquisition tools (mentioned in Section-II) in such a way that before each tool we restarted Windows operating system and *Karos online* game.

To analyze all memory dumps we used *Volatility Framework 2.4*, *WindowsSCOPE Ultimate*, and *Mandiant's Redline*.

C. Volatility Framework

Volatility framework is a command line open source volatile memory forensics analysis tool, commonly used for malware analysis and incident response. It is written in *Python* language under the GNU General Public License and is compatible with *Microsoft Windows*, *Linux* and *OS X*. *Volatility* supports almost all types of memory dump formats from 32 bit and 64 bit versions of operating systems. [6].

D. WindowsSCOPE Ultimate

WindowsSCOPE is a volatile memory forensics analysis and reverse engineering tool to analyze *Microsoft Windows* physical memory. It is also used for malware detection and its reverse engineering. It supports software based as well as hardware assisted techniques for both unlocked and locked systems. *WindowsSCOPE* displays results in user friendly format [7].

E. Mandiant's Redline

Redline is a *Mandiant's Premier* free tool, it offers host based investigative abilities to find malevolent activities through memory analysis. Users can systematically assemble all processes, drivers, network sockets, event logs, web history etc. available in memory at time of memory acquisition [8].

IV. Memory Acquisition Tools Examination

A. *Forensics Analysis of Scenario-I (no anti-debugging tool installed) with Volatility Framework*

As we know that Scenario-I is a normal scenario as there is no malicious application/rootkit installed. Only Windows 7 and some software like *Microsoft Word* and *Firefox* were installed. We analyzed all memory images acquired with under-examination tools with *Volatility framework 2.4*.

We applied three *Volatility Framework* plugins (*pslist*, *psscan* and *psxveiw*) on all acquired images. *pslist* walks the doubly link-list pointed to by PsActiveProcessHead and shows process name with its ID, its parent process ID, offset, date & time of creation etc. Some processes use pool scanning, i.e pool header, due to which *Volatility* cannot extract unlinked, hidden and all terminated processes with *pslist* plugin. To detect unlinked, hidden or inactive processes, *psscan* plugin can be used. Some rootkits can still hide themselves or other processes, but these types of hidden processes can be extracted from memory image with *psxview* plugin. *psxview* also compare PsActiveProcessHead linked list, EPROCESS and ETHREAD pool scanning, csrss.exe handles table and its internal linked list and PspCidTable [9]. In this scenario Volatility Framework successfully analyzed all memory images captured with under examined tools. Table-I shows summary of Volatility Framework results for all acquired images by applying *pslist*, *psscan* and *psxview* plugins.

B. *Forensics Analysis of Scenario-II (anti-debugging tool installed) with Volatility Framework*

In this scenario, we have analyzed all acquired memory images taken with under-examination tools with *Volatility Framework* and compared the results. Due to limitation of space, in this paper we only show the detail analysis of two images, taken with *Moonsols DumpIt* and *FTK Imager*. However, comparison of results acquired from all images is shown in Table-II.

1) Forensics Analysis of memory image taken with MoonSols DumpIt: As we already know about memory image basic information so we first run *kdbgscan* plugin to get advanced information of memory image i.e. detail about service pack, number of running processes, number of loaded modules in memory and information relevant to Windows kernel etc. The command line is given below and the output is shown in Fig 1.

E:\>volatility-2.4.exe --profile=Win7SP0x86 –f Windows7.raw kdbgscan

```
Volatility Foundation Volatility Framework 2.4
**************************************************
Instantiating KDBG using: Kernel AS Win7SP0x86 (6.1.7600 32bit)
Offset (V)                    : 0x82943be8
Offset (P)                    : 0x2943be8
KDBG owner tag check          : True
Profile suggestion (KDBGHeader): Win7SP0x86
Version64                     : 0x82943bc0 (Major: 15, Minor: 7600)
Service Pack (CmNtCSDVersion) : 0
Build string (NtBuildLab)     : 7600.16385.x86fre.win7_rtm.09071
PsActiveProcessHead           : 0x8295be98 (42 processes)
PsLoadedModuleList            : 0x82963810 (148 modules)
```

[snip]

Fig. 1 Output of kdbgscan plugin

TABLE-I: Results by Volatility Framework in Scenario-I

Results extracted by Volatility Framework in Scenario-I						
Memory Acquisition Tool (latest version)	***Volatility Framework 2.4 plugins***					
	Imageinfo	***Kdbgscan***	***Pslist***	***psscan***	***psxview***	***Remarks/ Comments about Tools***
MoonSols DumpIt	√	√	√	√	√	• Portable • Easy to use
Access Data FTK Imager	√	√	√	√	√	• Light version is portable • Dump redundant values oftentimes • GUI version causes maximum footprints
Winpmem 1.6.2	√	√	√	√	√	• Portable • easy to use Multiple options
Belkasoft RAM Capture	√	√	√	√	√	• Portable • Easy to use
Mandiant's Memoryze	√	√	√	√	√	• Not portable, needs to be installed on target system
Magnet RAM Capture	√	√	√	√	√	• Portable • Easy to use • Option to capture memory image in segments

Fig. 1 shows that 42 processes are running and 148 modules (code and data) are loaded in memory. Next we applied *pslist* plugin to extract information about running processes from memory image. The command line is given below and the output is shown in Fig 2.

E:\ >volatility-2.4.exe --profile=Win7SP0x86 –f Windows7.raw pslist

```
Volatility Foundation Volatility Framework 2.4

Offset(V)  Name            PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------- ----- ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0x84839ae0 System             4      0     76      525 ------      0 2015-04-23 09:32:29 UTC+0000
0x8584dd40 smss.exe         256      4      2       30 ------      0 2015-04-23 09:32:29 UTC+0000
0x85763d40 csrss.exe        344    332     12      471      0      0 2015-04-23 09:32:34 UTC+0000
0x85eb4d40 wininit.exe      392    332      4      100      0      0 2015-04-23 09:32:35 UTC+0000
0x85eacd40 csrss.exe        400    384     11      367      1      0 2015-04-23 09:32:35 UTC+0000
0x8591a350 services.exe     440    392      8      221      0      0 2015-04-23 09:32:35 UTC+0000
0x8513b380 lsass.exe        456    392      8      594      0      0 2015-04-23 09:32:35 UTC+0000
0x8513d568 lsm.exe          464    392     11      172      0      0 2015-04-23 09:32:35 UTC+0000
0x85ef6d40 winlogon.exe     496    384      4      132      1      0 2015-04-23 09:32:36 UTC+0000
0x8606c678 svchost.exe      784    440     21      473      0      0 2015-04-23 09:32:49 UTC+0000
0x8609a030 svchost.exe      844    440     23      671      0      0 2015-04-23 09:32:49 UTC+0000
0x861be818 explorer.exe    1552   1468     22      683      1      0 2015-04-23 09:32:53 UTC+0000
0x862ead40 Steam.exe       2020   1552     22      550      1      0 2015-04-23 09:32:55 UTC+0000
0x86271030 launchpoint.ex 2964   2020      0 --------      1      0 2015-04-23 09:34:28 UTC+0000 2015-04-23 09:35:05 UTC+0000
0x85fedbd0 svchost.exe     3372    440     14      391      0      0 2015-04-23 09:34:58 UTC+0000
0x858efc20 GameGuard.des   3948   1048      0 --------      1      0 2015-04-23 09:44:59 UTC+0000 2015-04-23 09:45:12 UTC+0000
0x84d40830 svchost.exe      964    440      5       81      0      0 2015-04-23 09:45:16 UTC+0000
0x84d674b8 DumpIt.exe      3320   2668      5       66      1      0 2015-04-23 09:45:51 UTC+0000
```

[snip]

Fig. 2 Output of pslist plugin

There are 42 processes extracted by *pslist* including three terminated process. Highlighted terminated processes (launchpoint.exe and gameguard.des.exe) are relevant to *Karos online* game that remained active in memory only for few seconds. As already mentioned, the memory image was taken when *Karos online* game (protected by *nProtect GameGaurd*) was running i.e. *Karos online* game application process must be loaded in memory, but this process is not shown in Fig. 2, it means that *Karos online* game application process was hidden by *nProtect GameGuard*.

To extract information about hidden processes we applied *psscan* plugin as *pslist* doesn't reveal hidden processes. The command line is given below and the output is shown in Fig 3.

E:\ >volatility-2.4.exe --profile=Win7SP0x86 -f Windows7.raw psscan

```
Volatility Foundation Volatility Framework 2.4

Offset(P)          Name              PID   PPID PDB        Time created                   Time exited
------------------ ---------------- ---- ------ ---------- ------------------------------ ------------------------------
0x000000007e271030 launchpoint.ex 2964   2020 0x7ec26480 2015-04-23 09:34:28 UTC+0000 2015-04-23 09:35:05 UTC+0000
0x000000007e2ead40 Steam.exe      2020   1552 0x7ec263c0 2015-04-23 09:32:55 UTC+0000
0x000000007e49c6c0 svchost.exe     872    440 0x7ec261c0 2015-04-23 09:32:49 UTC+0000
0x000000007e5ac030 svchost.exe    1496    440 0x7ec26300 2015-04-23 09:32:53 UTC+0000
0x000000007e5be818 explorer.exe   1552   1468 0x7ec26340 2015-04-23 09:32:53 UTC+0000
0x000000007e6acd40 csrss.exe       400    384 0x7ec26040 2015-04-23 09:32:35 UTC+0000
0x000000007e6b4d40 wininit.exe     392    332 0x7ec260a0 2015-04-23 09:32:35 UTC+0000
0x000000007e6f6d40 winlogon.exe    496    384 0x7ec26120 2015-04-23 09:32:36 UTC+0000
0x000000007e7edbd0 svchost.exe    3372    440 0x7ec264a0 2015-04-23 09:34:58 UTC+0000
0x000000007ec4dd40 smss.exe        256      4 0x7ec26020 2015-04-23 09:32:29 UTC+0000
0x000000007ecefc20 GameGuard.des  3948   1048 0x7ec262e0 2015-04-23 09:44:59 UTC+0000 2015-04-23 09:45:12 UTC+0000
0x000000007ed1a350 services.exe    440    392 0x7ec260c0 2015-04-23 09:32:35 UTC+0000
0x000000007ef63d40 csrss.exe       344    332 0x7ec26060 2015-04-23 09:32:34 UTC+0000
0x000000007f53b380 lsass.exe       456    392 0x7ec260e0 2015-04-23 09:32:35 UTC+0000
0x000000007f53d568 lsm.exe         464    392 0x7ec26100 2015-04-23 09:32:35 UTC+0000
0x000000007f809d40 GameMon.des    2348   1048 0x7ec264e0 2015-04-23 09:45:02 UTC+0000
0x000000007f88d030 AMo.exe        1048   3068 0x7ec26460 2015-04-23 09:44:58 UTC+0000
0x000000007f9674b8 DumpIt.exe     3320   2668 0x7ec265e0 2015-04-23 09:45:51 UTC+0000
0x000000007fc39ae0 System            4      0 0x00185000 2015-04-23 09:32:29 UTC+0000
```

[snip]

Fig. 3 Output of psscan plugin

There are 47 processes extracted by *psscan* plugin included 2 actual hidden processes (i.e., AMo.exe and GameMon.des), these hidden processes were not shown in task manger on live system and also couldn't be extracted by *pslist* plugin. Fig. 4 shows parent-child relationship of AMo.exe with GameMon.des and GameGaurd.des processes.

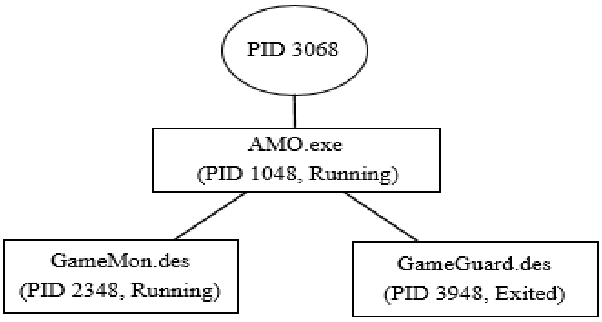


Fig. 4 Child processes of AMo.exe

Actually AMo.exe (PID 1048) is *Karo online* game application process which was hidden by *nProtect GameGaurd*. AMo.exe had created two of its child processes, GameMon.des which was running but hidden process and GameGuard.des which was exited after game lunches successfully. This process provides protection to game launching process from crashing. GameMon.exe process monitors running game and runs as long as game application is running. "*nProtect GameGuard*" concealed AMo.exe and

GameMon.des processes to defend them from cheating and bugging activities/ applications.

Fig. 3 & 4 shows that parent process ID of AMo.exe process is "3068" but on analysing memory image for this process with Volatility Framework we found that this process was not active in memory at the time of memory acquisition. We analyzed running *Karos online* game on live machine with Process Monitor and Process Explorer tools (included in Microsoft Sysinternals Suite), we found that AMo.exe process was created by LaunchKaros.exe process, which exited permanently when *Karos online* games successfully started, hence PID 3068 was associated with Launchkaros.exe. Fig. 5 shows the Karos online game processes hierarchy.

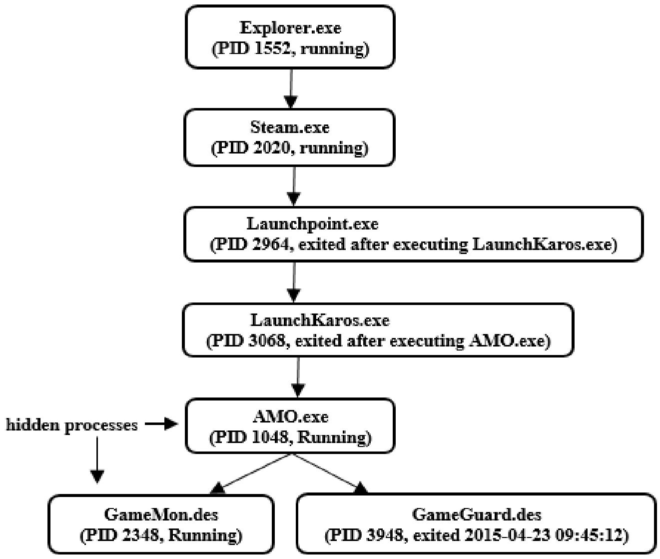


Fig. 5 Karos online game process structure in memory

Steam.exe is a child process of explorer.exe, it created launchppoint.exe process which itself exited after starting launchkaros.exe. LaunchKaros.exe started AMo.exe process and then immediately exited when game application successfully started. *nProtect GameGaurd* concealed AMo.exe immediately when game started and kept it hidden during the entire session. "*psxview*" plugin can be used to extract all terminated and hidden processes but we didn't require to show result of *psxview* plugin because we already got detail of hidden processes relevant to Karos online game by applying *psscan* plugin.

2) Forensics Analysis of memory image taken with Access Data FTK Imager: We run *kdbgscan* plugin but we got blank output which shows that FTK imager couldn't capture any information such as details about service pack, number of running processes, number of loaded modules in memory etc. In this case the anti-debugging and anti-cheating software was running.

In next command we applied *pslist* plugin in order to extract running process. The command line is given below and the output is shown in Fig 7.

E:\ >volatility-2.4.exe --profile=Win7SP0x86 -f memdump.mem pslist

```
Volatility Foundation Volatility Framework 2.4
No suitable address space mapping found
Tried to open image as:
 MachOAddressSpace: mac: need base
 LimeAddressSpace: lime: need base
 WindowsHiberFileSpace32: No base Address Space
 WindowsCrashDumpSpace64BitMap: No base Address Space
 WindowsCrashDumpSpace64: No base Address Space
 [snip]
```

Fig. 7 Output of pslist

We didn't get any detail about running processes by applying *pslist* plugin. Fig. 7 indicates that FTK imager can't dump running processes when *nProtect GameGuard* (anti-debugging tool) is enabled. By using *psscan* and *psxview* plugins, we got blank output values as shown in Fig. 8 and Fig. 9 respectively.

E:\ >volatility-2.4.exe --profile=Win7SP0x86 -f memdump.mem psscan

```
Volatility Foundation Volatility Framework 2.4
Offset(P)  Name    PID  PPID PDB    Time created    Time exited
------------------ ---------------- ------ ------ ---------- ---
```

Fig. 8 Output of psscan plugin

E:\ >volatility-2.4.exe --profile=Win7SP0x86 -f memdump.mem psxview

```
Volatility Foundation Volatility Framework 2.4
Offset(P)  Name   PID pslist psscan thrdproc pspcid csrss session deskthrd ExitTime
--------- ------------------ ------ ------ ------ -------- ------ ----- ------- -
```

Fig. 9 Output of psxview plugin

By analysing results extracted by Volatility Framework from memory image captured with FTK Imager we can determined that FTK Imager can't dump Windows volatile memory reliably in normal scenario and unable to dump memory image(specially kernel space artifacts) when there is some rootkit or anti-debugging process running on the target system. We have also captured volatile memory of different version of Windows operating system and analyzed it with different memory analysis tools but we couldn't get reliable results as compared to memory dumps taken with other tools like Moonsols DumpIt, Belkasoft Ram Capture, Memoryze etc.

Another acquisition tool, *Magnet RAM Capture* also shows similar behaviour like FTK Imager. Magnet RAM capture can work when there is no anti-debugging or anti-cheating application installed on target system, but it cannot dump memory completely (user and kernel space) and accurately when advanced rootkits are installed on the system and, fail to dump Windows volatile memory when anti-debugging program is running in target system.

In contrast, all other tools like MoonSols DumpIt, Belkasoft RAM Capture, Mandiant's Memoryze and Winpmem captured Windows volatile memory accurately which were proven when we analyzed these memory images with popular memory analysis tools.

The results of Volatility Framework from memory images taken with under-examination tools of Scenario-II are displayed in Table-II.

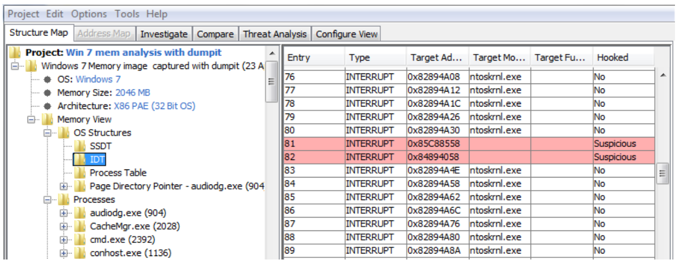
C. *Forensics Analysis (anti-debugging tool installed) with WindowsSCOPE – Ultimate*

1) Forensics Analysis of memory image taken with MoonSols DumpIt and FTK Imager: WindowsSCOPE is GUI tool for Windows memory analysis. It can extract Windows OS structure such as SSDT (System Service Descriptor Table), Interrupt Descriptor Table (IDT), and Process Table with modules, Page Directory Pointer for each process, drivers, open files, open network sockets and open registry keys etc. We have analyzed all memory images taken with under examination tools.

TABLE-II: Results by Volatility Framework in Scenario-II

Results extracted by Volatility Framework in Scenario-II						
Memory Acquisition Tool (latest version)	***Volatility Framework 2.4 plugins***					
	imageinfo	***Kdbgscan***	***Pslist***	***psscan***	***psxview***	***Remarks/ Comments***
MoonSols DumpIt	√	√	√	√	√	Dump both user & kernel space data
Access Data FTK Imager	×	×	×	×	×	Unable to dump user/kernel space data accurately in anti-debugging/ anti-cheating environment
Winpmem	√	√	√	√	√	Dump both user & kernel space data
Belkasoft RAM Capture	√	√	√	√	√	Dump both user & kernel space data
Mandiant's Memoryze	√	√	√	√	√	Dump both user & kernel space data
Magnet RAM Capture	×	×	×	×	×	Unable dump user/kernel space data accurately in anti-debugging/ anti-cheating environment

Fig. 10 shows the results, analyzed and extracted by WindowsSCOPE from Windows 7 memory image taken with MoonSols DumpIt.



[snip]

Fig. 10 Extracted artifacts with WindowsSCOPE from Memory image captured with MoonSols DumpIt

WindowsSCOPE successfully extracts all artifacts (claimed by WindowSCOPE - Ultimate) from memory image captured with DumpIt. Fig. 10 is an interface of results extracted by WindowsSCOPE from memory image captured with DumpIt.

Memory images captured from systems in which no anti-debugging or anti-cheating is enabled (or installed) with under examined acquisition tools were successfully analyzed by WindowsSCOPE. But it was unsuccessful to analyze memory images captured with FTK imager and Magnet RAM Capture from target system in which anti-debugging & anti-cheating application (i.e *nProtectGameGaurd)* was installed. Table-III shows results of WindowsSCOPE memory analysis for all images captured with under examined tools.

TABLE-III: Results by WindowsSCOPE for all memory images (anti-debugging installed on target system) in Scenario-II

Results extracted by WindowsSCOPE for all memory images (anti-debugging tool installed on target system)								
Memory Acquisition Tool (latest version)	***WindowsSCOPE features***							
	SSDT	***IDT***	***Processes***	***Page Table Entries***	***drivers***	***Open Files***	***Network Sockets***	***Registry Keys***
MoonSols DumpIt	√	√	√	√	√	√	√	√
Access Data FTK Imager	×	×	×	×	×	×	×	×
Winpmem	√	√	√	√	√	√	√	√
Belkasoft RAM Capture	√	√	√	√	√	√	√	√
Mandiant's Memoryze	√	√	√	√	√	√	√	√
Magnet RAM Capture	×	×	×	×	×	×	×	×

Table –III shows the results which we got by analyzing captured memory images with WindowsSCOPE. In above table results of FTK Imager and Magnet RAM Capture proves that both these memory capturing tools are unable to dump memory accurately when anti-debugging program is running.

D. *Forensics Analysis (anti-debugging tool installed) with Mandiant's Redline.*

Mandiant's Redline can also extract running processes, drivers, network connections, open registry keys etc., it also has ability to analyze suspicious processes.

We analyzed all Windows 7 memory images captured with Redline in both scenarios. In Scenario-I in which no anti-debugging tool was installed on target system, it successfully analyzed all images without any error. However, in Scenario-II in which anti-bugging tool was installed on target system, we got errors in analysis of some memory images with Redline. These errors are discussed below:-

1) Forensics Analysis of memory image taken with FTK Imager and Magnet RAM Capture: Redline analyzed both images captured with FTK Imager captured and Magnet RAM, but it couldn't find operating system version and processes. Fig. 11 is message displayed by Redline after spending too much time on analysis of memory images captured with Access Data FTK Imager and Magnet RAM Capture.

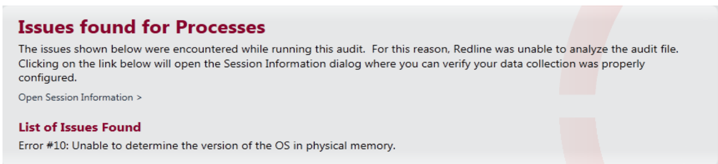
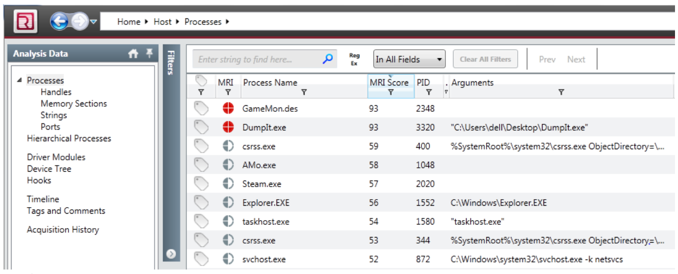


Fig. 11 Redline output message after analyzing memory images (Captured with FTK Imager & Magnet RAM Capture)

2) Forensics Analysis of memory image taken with MoonSols DumpIt, Belkasoft RAM Capture, Memoryze and Winpmem:

We analyzed memory image captured with DumpIt, Belkasoft RAM Capture, Memoryze and Winpmem in

Scenario-II (anti-debugging tool installed) with Redline. Redline analyzed these images very efficiently. It also extracted hidden processes (AMO.exe, GameMon.des etc.) from these images and showed result in user friendly interface. Artifacts extracted by Redline from memory image captured with DumpIt are shown in Fig. 12.



[snip]

Fig. 12 Redline results extracted from memory image captured with DumpIt

TABLE-IV: Results by Mandiant's Redline for all memory images (anti-debugging installed on target system) in Scenario-II

Results extracted by Mandiant's Redline for all memory images (anti-debugging installed on target system)								
Memory Acquisition Tool (latest version)	***Mandiant's Redline features***							
	Processes	***Driver Modules***	***Device Tree***	***SSDT***	***IDT***	***drivers***	***Registry keys***	***Network Sockets***
MoonSols DumpIt	√	√	√	√	√	√	√	√
Access Data FTK Imager	×	×	×	×	×	×	×	×
Winpmem	√	√	√	√	√	√	√	√
Belkasoft RAM Capture	√	√	√	√	√	√	√	√
Mandiant's Memoryze	√	√	√	√	√	√	√	√
Magnet RAM Capture	×	×	×	×	×	×	×	×

V. Result Comparison

We have compared popular Windows memory acquisition tools through popular Windows memory forensics analysis software. This research shows that not every tool is capable to acquire physical memory contents accurately, if system is infected by malware or protected by anti-debugging programs and in some cases tools are unable to dump volatile memory when anti-debugging programs are running in target systems. By capturing Windows 7 memory in presence of running *Karos online* game along with *nProtect GameGaurd* (ani-debugging & anti-cheating tool), we have investigated that some Windows memory acquisition tools couldn't acquire Windows volatile memory accurately when target system is being protected by anti-cheating or anti-debugging programs or infected by sophisticated rootkits. Table-V shows comparison of our analysis for different Windows memory acquisition tools with respect to their memory acquisition capabilities.

TABLE-V: Comparison of Windows Memory Acquisition Tools

Comparison of Memory Acquisition Tools via Memory Analysis Tools				
Memory Acquisition Tool (latest version)	***Clean Target System (no anti- debugging application installed/running)***		***Suspicious Target System(anti-debugging application running)***	
	Memory Capturing Features Accuracy		***Memory Capturing Features Accuracy***	
	User Space	***Kernel Space***	***User Space***	***Kernel Space***
MoonSols DumpIt	√	√	√	√
Access Data FTK Imager	√	√	×	×
Winpmem 1.6.2	√	√	√	√
Belkasoft RAM Capture	√	√	√	√
Mandiant's Memoryze	√	√	√	√
Magnet RAM Capture	√	√	×	×

VI. Conclusion

The goal of this research work has been to compare accuracy and strength of different available Windows memory acquisition tools, because for memory forensics analysis, it is very important that the memory of target system is captured properly i.e. the tool must capture memory accurately and completely from both user and kernel areas of memory of target system. This research has conducted a detailed analysis of memory captured under different scenarios with different tools and has compared the accuracy and completeness of memory acquisition by these tools. The results show that it is possible for processes to deny complete and accurate acquisition of memory to the tools. Further, the results can also be used by investigators to choose a suitable memory acquisition tool for their investigation.

References

[1] Michael Hale Ligh, Andrew Case, Jamie Levy and AAron Walters, "The Art of Memory Forensics", http://www.memoryanalysis.net/#!amf/cmg5.

[2] Bradley Schatz, "BodySnatcher: Towards reliable volatile memory acquisition by software", http://www.sciencedirect.com.

[3] Tribble Grand Idea Studio, http://www.grandideastudio.com /po rtfolio/ tribble/.

[4] Liming Cai, Jing Sha and Wei Qian," Study on Forensic Analysis of Physical Memory", (3CA 2013).

[5] nProtect GameGaurd, http://www.inca.co.kr/, http://www.npr otect.com/v7/b2b/sub.html?mode=game.

[6] The Volatility, http://github.com/volatility foundation/volatility.

[7] WindowsSCOPE – Ultimate, http://www.windowsscope.com.

[8] Mandiant's Redline, https://www.mandiant.com/.

[9] Volatility Commands for Windows Malware, https:/code.google. com/p/volatility/ wiki/CommandReferenceMal23#psxview.