A Study: Volatility Forensic on Hidden Files

Cutifa Safitri[1]

[1]Department of Computer Science, International Islamic University Of Malaysia, 53100 Kuala Lumpur, Malaysia

Abstract: ***More and more forensic researchers gain findings by live investigating memory volatility. Forensic research on volatile data is nowadays big area of interest. In the early days, investigators pulled the plug, but now it could be very interesting to capture the volatile data of the system. Memory forensics and data carving are among methods that are usually used during volatile investigation. Before pulled the plug, it is often desirable to capture volatile information that may not be recorded in a file system or image backup, such as processes and the contents of memory. This data may hold clues as to the attacker's identity or the attack methods that were used. However, risks are associated with acquiring information from the live system. Any action performed on the host itself will alter the state of the machine to some extent. In paper, an analysis of hidden process on volatility is conducted.***

Keywords: RAM, Forensic, Volatility, Hidden Files**.**

1. Introduction

Forensic research of volatile data is nowadays a big area of interest. In the early days investigators often pulled the plug of the evidence PC, but now it could be very interesting to capture the volatile data of a system. Memory forensics and data carving are methods that are more and more used during investigations [1].

Before pulled the plug, it is often desirable to capture volatile information that may not be recorded in a file system or image backup, such as processes and the contents of memory. This data may hold clues as to the attacker's identity or the attack methods that were used. However, risks are associated with acquiring information from the live system. Any action performed on the host itself will alter the state of the machine to some extent [2].

The volatile data is referred to as stateful information from the subject system while it remains powered on. In this project, the value of volatile data is not limited to process memory associated with malware, but can include passwords, Internet Protocol (IP) addresses, Security Event Log entries, and either contextual details that can provide a more complete understanding of the malware and its use on a system.

In forensic methodology, the investigator must ensure that enough information is collected to determine whether or not an incident has occurred and should be sure to collect as much volatile information as possible. The investigator also must ensure that a minimum of activity will occur on the "victim" system (reducing the changes made to the system and the "footprint" left by the investigator or first responder), ensure that data will not be written to the system, and potentially untrusted executables on the system will not be run [3]. If the investigator can capture the image on what happened on that current time, it will lead the investigator on what happened on that particular time. Since the size of RAM keep increasing until up to 8 Gigabytes recently, the attacker would not need hard disk as virtual memory to run any malicious program on its targeted machine. It will be all handled by the RAM. Now, the problem is what happened to these hidden processes?

Malware send data to extension site and create a backdoor. Some network open list tools such as pslist and netcat could not detect the open network that created by malware. It captures data from volatility memory which includes data on unallocated space on RAM. Malware is always being customized to avoid detection, to become bullet proof from anti-viruses and to find the file that created by the mal-code. The objective of this project is to reveal the hidden data that might be caused by a malware, which intended to be hid by the attacker.

The benefit of this research is to strengthen an investigation. This is also to make aware that volatility is one of the critical parts of an investigation. The resources of this project investigation are supported by certified security investigation.

2. Problem Description

Each process has their structure on the kernel's memory which called executive process. This executive process (EPROCESS) contains much information, such us the process name, the process ID name, the exe name and many more. The EPROCESS work like a doubly linked list. The Flink member of this structure points to the next entry process while the Blink member points to the previous entry (process).

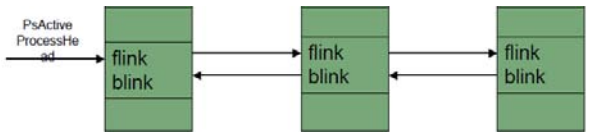


Figure 1: FLink and Blink

The EPROCESS of the current running program will reside inside RAM. The problem will arise when there is a hidden process on RAM image that the investigator collected. Could that process be found? When a process is being hid, it disconnected from the doubly-linked list.

Figure 2: Broken Flink and Blink

Thus, when a program is listing processes, it skips over the one that the attacker hid. This kind of technique is commonly used by rootkits to conceal their processes. The objective is to find the hidden process and gather the information. The findings should answer why those processes is being hid in the first place.

3. Literature Review and Case Study

Forensic is becoming an important part in the security field; evidence of the related crime is spread across the drive. Some attacker prefer to reside the evidence in the ram memory since its volatile, thus the evidence is gone when the machine power is off.

Forensic is growing and find a way to retrieve information before the machine turned off. A live digital forensic is conducted to retrieve volatile information. But studies on the tools used have told that different investigator has different point of view and the tools used is also vary. A comparison on the tools has shown that some tools make major changes in every keystroke.

Again, forensic is growing into digital forensics in virtual environment. The idea is to create a virtual machine from an image of the machine to be investigated. Some tools are developed to support this investigation. This approach is considered powerful but takes a lot of additional resources since virtual computer uses real hardware components present in the computer it runs on. Thus, recent research comes out with offline analysis, the ram forensic.

As the RAM is getting bigger and bigger, many files and process running is stored inside the RAM. Memory inside computer is becoming targeted place to run malicious files. We cannot imagine what are the things stored inside our memory. All the important sources and current state of the machine is all stored in the single memory. Imagine if we dump the whole memory including BIOS memory, an investigator can predict what did happened during incident. Based on the current information demand, I am focusing my project into volatility offline analysis.

RAM forensic start from dumping the whole memory and conduct offline analysis based on the information gathered from the synopsis report and the unusual things happened in the machine [4]. Volatility has broad libraries to detect evidence from the image. Start from the malware detection, open port, files accessed, keyboard buffer and encryption files.

Since every tool has their function, this paper would like to propose a literature to detect any hidden file that the attacker hides after the incident. The importance of the volatility in hidden files is to reveal the facts and give relevant information for hard disk investigation.

After go through any reliable sources and read the past literature, this project is fits under digital forensic media and tool expansion [5]. The growing volatility tools is Volatility Framework [6], it is one of the largest open source project for digital forensic. The tool was build using python and already provides a python libraries developed by many forensic developer tools [7]. It is mainly used for the extraction of digital artifacts from RAM [8]. There are several capabilities offered by this tool.

3.1 Scope

The paper focus on the way of findings the hidden process in memory forensic, thus a guideline will be made. The findings process is conducted to retrieve as much as possible information from the PC.

3.2 Resource Requirements

The resource requirements that are required in this paper are:

1. A computer for developing and running the application. Personal Computer (PC) is used to develop and run the application. In the system PC, the main application will be installed.
2. Microsoft XP Service Pack 2 as the operating system. This software is an operating system to support the development and implementation of the main application.
3. LiveKD, it allows to run the Kd and Windbg Microsoft kernel debuggers, which are part of the Debugging Tools for Windows package, locally on a live system

3.3 System Requirements

System requirement; general requirement regarding the concept, performance and interface, required in this paper are:

1. The PC is used and acts as the infected PC, which will be investigated.
2. Enable to grab the current process.
3. Provide and ensure to list the hidden process to the investigator.
4. Display information about the hidden process.
5. Provide the guideline on how to get the hidden process.
6. Information providing: the guideline must provide step by step information about how to get the hidden process, and let the investigator acquires the information they need. The investigator can retrieve information by sending command through the *livekd*

3.4 System Architecture

In this paper, the system architecture is use to assist and complete the flow on how the student way of thinking about this project. The architecture uses are:

1. Case diagram: Provide a graphical overview of the functionality provided by a system in terms of actor, their goals and any dependencies between those use cases.
2. Activity diagram: Is a loosely defined diagram technique for showing workflows of stepwise activities and actions, with support for choice, iteration and concurrency.

4. Discussion

The memory forensic study is divided into two main category (or perspectives) which are the affected PC and the investigator. To complete the scenario, some memory image containing malware is needed.

For the environment for the affected PC, there are some of software needs to be installed in this environment. The Windows XPSP2 is installed inside the QEMU Manager. QEMU is a popular emulation environment for Windows and Linux.

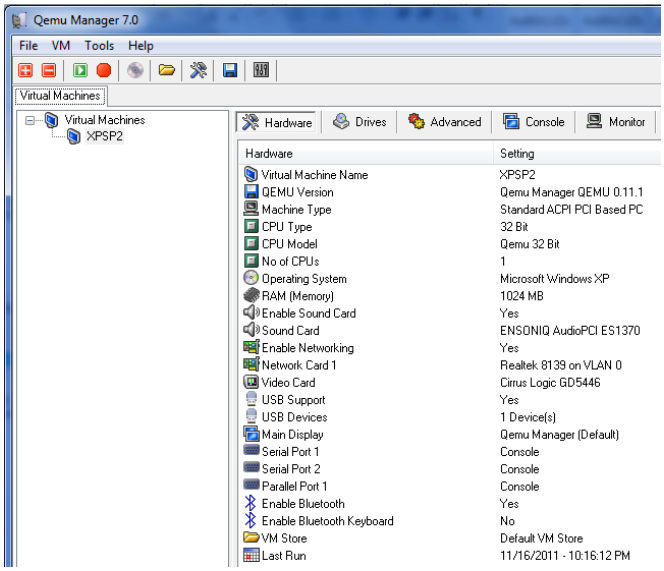


Figure 3: QEMU Manager

Inside this QEMU Manager, the livekd is execute all the debugger commands that work on crash dump files to look deep inside the system.

This subsection describes list of the commands that will be executed by investigator with using livekd. The result from each command is displayed with the figures along with the description for each result. The flow chart below simplifies the action taken by the investigator in order to find the hidden process.

The second and most important environment is the investigator's environment. Following will describes list of the commands that will be executed by investigator with using livekd. The result from each command is displayed with the figures along with the description for each result. The flow chart beside simplifies the action taken by the investigator in order to find the hidden process.

4.1 List all loaded objects

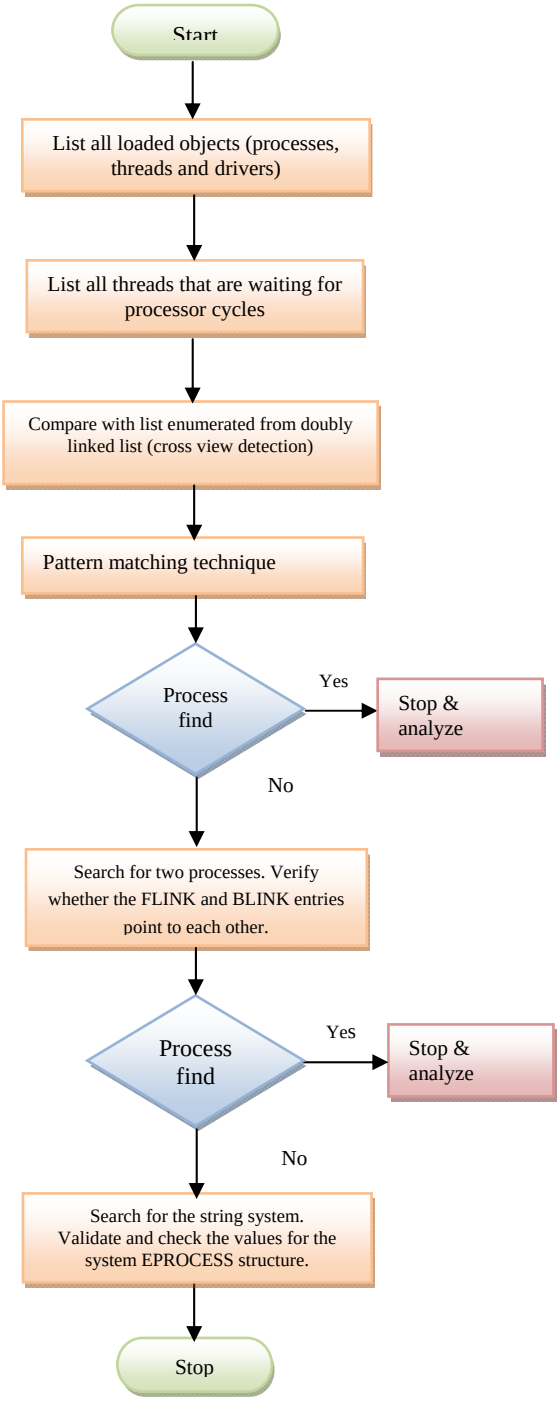
Using the first step in the flow chart, first we will list all loaded objects such as processes, threads and drivers. By the kernel debugger, we can display the process list with command:

```
kd > !process 0 0
```

The list of the current process running will appear with its Cid. By grab the Cid numbers we may get the detail of the process block.

```
kd> !process 0 0
**** NT ACTIVE PROCESS DUMP ****
PROCESS 867c6a00  SessionId: none  Cid: 0004    Peb: 00000000  ParentCid: 0000
    DirBase: 00039000  ObjectTable: e1001d28  HandleCount: 242.
    Image: System

PROCESS 8660b020  SessionId: none  Cid: 0140    Peb: 7ffd6000  ParentCid: 0004
    DirBase: 0ad99000  ObjectTable: e100c818  HandleCount:  21.
    Image: smss.exe

PROCESS 86789c38  SessionId: 0  Cid: 01e4    Peb: 7ffd6000  ParentCid: 0140
    DirBase: 0b86e000  ObjectTable: e1009120  HandleCount: 325.
    Image: csrss.exe

PROCESS 865a1900  SessionId: 0  Cid: 01fc    Peb: 7ffdc000  ParentCid: 0140
    DirBase: 0b973000  ObjectTable: e13951e0  HandleCount: 506.
    Image: winlogon.exe

PROCESS 86589318  SessionId: 0  Cid: 022c    Peb: 7ffdf000  ParentCid: 01fc
    DirBase: 0ba3f000  ObjectTable: e148e710  HandleCount: 259.
    Image: services.exe

PROCESS 865868e8  SessionId: 0  Cid: 0238    Peb: 7ffdd000  ParentCid: 01fc
    DirBase: 0baca000  ObjectTable: e16188d0  HandleCount: 336.
    Image: lsass.exe

PROCESS 8657dda0  SessionId: 0  Cid: 02d4    Peb: 7ffde000  ParentCid: 022c
    DirBase: 0bd49000  ObjectTable: e161fee8  HandleCount: 188.
    Image: svchost.exe

PROCESS 8656ada0  SessionId: 0  Cid: 0314    Peb: 7ffd9000  ParentCid: 022c
    DirBase: 0be84000  ObjectTable: e138c658  HandleCount: 254.
```

Figure 4: List all loaded objects



Flow Chart: Investigator Guideline

4.2 List all threads that are waiting for processor cycles

In the contents of the Executive Thread Block, one of the key content is ALPC information, which contents message ID that the thread is waiting for and address of message. The kernel debugger command below dumps a subset of the information in the thread data structures [3].

```
kd > !thread
```

Some key elements of the information displayed by the kernel debugger cannot be displayed by any utility. In this case, for threads in a wait state, the list of objects the threads is waiting for.

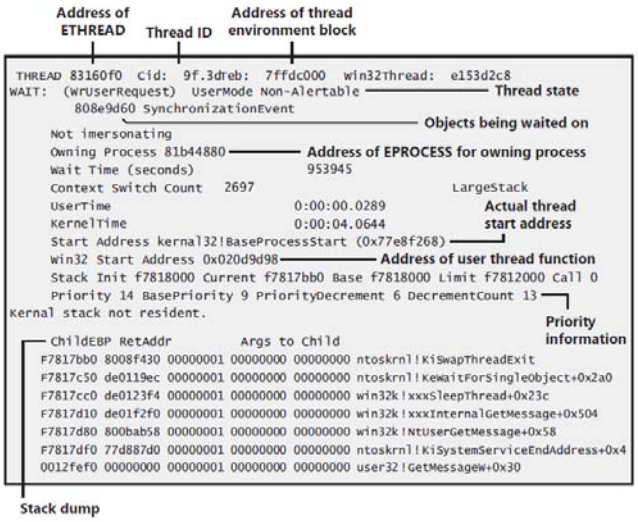


Figure 5:!thread

4.3 Compare with list enumerated from doubly linked list

If there are no idle processors when a thread wants to run, Windows compares the propriety of the thread running (or the one in the standby state) on the thread's ideal processors to determine whether it should preempt that thread.

4.4 Pattern matching technique

Now we already on the fourth step of the flow chart. On the first step we already list all loaded objects by using command:!process 0 0 Using that command giving us the details of all processes runs, in this pattern matching techniques two important details from the processes is the CID and PEB, which we will using it now.

```
kd > !process [with the cid]
```

By run this command we will get the detail of the process block, along with their token id. As pictured in the output below, the process running with using Cid 0190, which belongs of processes names cmd.exe. In the end of the process, it displayed "not impersonating". It is a clue since a thread can assume a different security context than that of its process. This mechanism is called impersonation.

When a thread is impersonating, security validation mechanisms use the thread's security context instead of that of the thread's process. When a thread isn't impersonating, security validation falls back on using the security context of the thread's owning process.

It's important to keep in mind that all the threads in a process share the same handle table, so when a thread opens an object-even if it's impersonating-all the threads of the process have access to the object.

```
PROCESS 862d2020  SessionId: 0  Cid: 0190    Peb: 7ffdd000  ParentCid: 07d4
    DirBase: 103a2000  ObjectTable: e1140c28  HandleCount:  31.
    Image: cmd.exe

PROCESS 862dd918  SessionId: 0  Cid: 0350    Peb: 7ffda000  ParentCid: 0190
    DirBase: 29e3e000  ObjectTable: e229d480  HandleCount:  30.
    Image: livekd.exe

PROCESS 86480700  SessionId: 0  Cid: 0354    Peb: 7ffd8000  ParentCid: 0350
    DirBase: 2a2d6000  ObjectTable: e21f95f8  HandleCount: 114.
    Image: windbg.exe

kd> !process 0190
Searching for Process with Cid == 190
Cid Handle table at e1003000 with 306 Entries in use
PROCESS 862d2020  SessionId: 0  Cid: 0190    Peb: 7ffdd000  ParentCid: 07d4
    DirBase: 103a2000  ObjectTable: e1140c28  HandleCount:  31.
    Image: cmd.exe
    VadRoot 8642d6c0 Vads 53 Clone 0 Private 128. Modified 0. Locked 0.
    DeviceMap e1a31820
    Token                             e21d94e0
    ElapsedTime                       00:00:52.328
    UserTime                          00:00:00.187
    KernelTime                        00:00:00.375
    QuotaPoolUsage[PagedPool]         31316
    QuotaPoolUsage[NonPagedPool]      2120
    Working Set Sizes (now,min,max)  (557, 50, 345) (2228KB, 200KB, 1380KB)
    PeakWorkingSetSize                557
    VirtualSize                       29 Mb
    PeakVirtualSize                   35 Mb
    PageFaultCount                    585
    MemoryPriority                    BACKGROUND
    BasePriority                      8
    CommitCharge                      472

        THREAD 864795b8  Cid 0190.007c  Teb: 7ffdf000 Win32Thread: e21c0968 WAIT: (UserRequest) U
            862dd918  ProcessObject
        Not impersonating
```

Figure 6: Process with CID 0190

After we run this command, another detail that we got is the token id. As pictured above, the cmd.exe has a token id which is e21d94e0. To look into the token details, we can run the command below:

```
kd > !token [with the token id]
```

The Security reference monitor (SRM) uses an object called a token (or access token) to identify the security context of a process or thread. A security context consists of information that describes the privileges, accounts, and groups associated with the process or thread.

The security mechanisms in Windows use two token components to determine what objects can be accessed and what secure operations can be performed. One component comprises the token's user account SID and group SID fields. The SRM uses SIDs to determine whether a process or thread can obtain requested access to a securable object, such as an NTFS file.
The second component in a token that determines what the token's thread or process can do is the privilege array. A token's privilege array is a list of rights associated with the token. An example privilege is the right for the process or thread associated with the token to shut down the computer.

By including security information in tokens, Windows makes it convenient for a process or thread to create objects with standard security attributes, because the process or thread doesn't need to request discrete security information for every object it creates.

The investigator can indirectly view token contents with Process Explorer's security tab of its process properties dialog box. The dialog box shows the groups and privileges included in the token of the process the investigator examine.

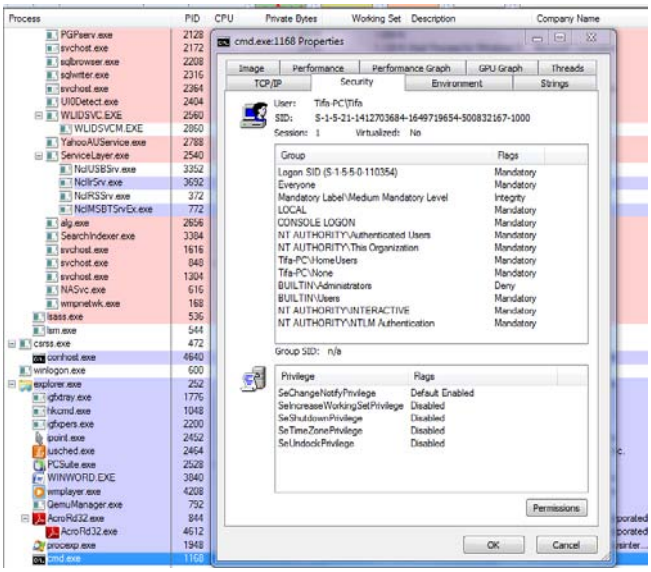


Figure 7: GUI for Token Credentials

The last step of the pattern matching is by checking the process Process Environment Block. The PEB is interesting to attackers is because the memory pages where the PEB resides are both Writeable and eXecutable. Since the PEB has always been in a fixed location, attackers have the option of copying their shellcode to a known address in the PEB and then transferring execution to that location. This technique can be useful when an arbitrary memory overwrite is possible but the payload is difficult to locate – such as for heap overflows.

Last time the kernel debugger! Process command displays a subset of the information in an EPROCESS block. Take the PEB value and run the following command:

```
kd > !peb [with the peb id]
```

It will give result of the ImageBaseAddress which contain base address of section, and the ProcessHeaps which is the first byte after PEB.

We will use the example of the windbg.exe; it has peb value of 7ffd8000.

```
PROCESS 86480700  SessionId: 0  Cid: 0354    Peb: 7ffd8000  ParentCid: 0350
    DirBase: 2a2d6000  ObjectTable: e21f95f8  HandleCount: 114.
    Image: windbg.exe
```

Figure 8: The !PEB value for 7ffd8000.

Take the peb value will give us:

```
kd> !peb 7ffd8000
PEB at 7ffd8000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            No
    ImageBaseAddress:         01000000
    Ldr                       00191e90
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 00191f28 . 00192b40
    Ldr.InLoadOrderModuleList:           00191ec0 . 00192b30
    Ldr.InMemoryOrderModuleList:         00191ec8 . 00192b38
            Base TimeStamp                     Module
         1000000 49a5f6a7 Feb 26 09:55:51 2009 C:\program files\Debugging Tools for Windows (x86)
        7c900000 411096b4 Aug 04 15:56:36 2004 C:\WINDOWS\system32\ntdll.dll
        7c800000 411096b4 Aug 04 15:56:36 2004 C:\WINDOWS\system32\kernel32.dll
        77dd0000 411096a7 Aug 04 15:56:23 2004 C:\WINDOWS\system32\ADVAPI32.dll
        77e70000 411096ae Aug 04 15:56:30 2004 C:\WINDOWS\system32\RPCRT4.dll
        77f10000 41109697 Aug 04 15:56:07 2004 C:\WINDOWS\system32\GDI32.dll
        77d40000 411096b8 Aug 04 15:56:40 2004 C:\WINDOWS\system32\USER32.dll
        77c10000 41109752 Aug 04 15:59:14 2004 C:\WINDOWS\system32\msvcrt.dll
         2000000 49a5f69f Feb 26 09:55:43 2009 C:\program files\Debugging Tools for Windows (x86)
         3000000 49a5f692 Feb 26 09:55:30 2009 C:\program files\Debugging Tools for Windows (x86)
        77c00000 411096b7 Aug 04 15:56:39 2004 C:\WINDOWS\system32\VERSION.dll
        774e0000 411096f2 Aug 04 15:57:38 2004 C:\WINDOWS\system32\ole32.dll
        7c9c0000 411096b7 Aug 04 15:56:39 2004 C:\WINDOWS\system32\SHELL32.dll
        77f60000 411096bc Aug 04 15:56:44 2004 C:\WINDOWS\system32\SHLWAPI.dll
        773d0000 4110968c Aug 04 15:55:56 2004 C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Con
        71b20000 411096be Aug 04 15:56:46 2004 C:\WINDOWS\system32\MPR.dll
        5ad70000 411096bb Aug 04 15:56:43 2004 C:\WINDOWS\system32\uxtheme.dll
        4b400000 41109708 Aug 04 15:58:00 2004 C:\WINDOWS\system32\MSFTEDIT.DLL
          1d00000 49a5f6a6 Feb 26 09:55:50 2009 C:\program files\Debugging Tools for Windows (x86)
    SubSystemData:     00000000
    ProcessHeap:       00090000
    ProcessParameters: 00020000
    WindowTitle:  'livekd.exe -w'
```

Figure 9: The !PEB result

5. Conclusion and Future Enhancement

The first step in responding to an incident is to have a policy for doing so. A corporate security policy that addresses how incidents will be handled is extremely important because it provides a roadmap for investigators. In some cases, the investigator may pull the plug and image the whole drive then start the investigation. This option, however, requires a great deal of interaction from the investigator- documenting commands, starting and restarting the PC, ensuring that all of the necessary commands are run when copying files, etc. Other option exist, such a more forensically sound methodology called volatility collection. As a computer program will get the process run off of the system after unplugging the electricity, this methodology gives additional and critical information to the investigator, including hidden files. Additionally, this methodology also allows for all of the necessary and significant data to be collected quickly, while decreasing the potential for mistakes.

Regardless of the methodology used, the goal should be the same. When investigating an incident, the primary focus should be to determine what happened through the collection and analysis facts. Filling in gaps by speculation can be just as useless as if an investigation had not been done at all.

References

[1] Beek, C. "Virtual Forensics." Ten ICT the Professionals.

[2] Scarfone, K., T. Grance, et al. (2008). Computer Security Incident Handling Guide.

[3] Carvey, H. (2005). Windows Forensics and Incident Recovery, Addsion Weasley.

[4] Russinovich, M. E. and D. A.Solomon (2010). Windows Internals Microsoft.

[5] Recommendations of the National Institute of Standards and Technology. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, MD 20899-8930, NIST SPecial Publication 800-61.

[6] The Volatility Framework: Volatile memory artifact extraction utility framework, application reftreived from https://www.volatilesystems.com/default/volatility.

[7] Technical White Paper Generic Anti-Exploitation Technology for Windows. e. D. Security.

[8] TMurgent, T. (2003). "Processor Affinity Multiple CPU Scheduling."

Author Profile

Cutifa Safitri received her B.CS (Honored) from International Islamic University Malaysia in 2011 and currently enrolled in M.IT at Kulliyyah of Information and Communication Technology, International Islamic University Malaysia.