|  |
| --- |
|  |
| Capstone Experience  IST 894 |
| Lab 12 – Windows Memory Forensics  Scott Finlon |

Table of Contents

[1. General Context 3](#_Toc88669260)

[2. Technical Context 4](#_Toc88669261)

[3. Solution 5](#_Toc88669262)

[References 13](#_Toc88669263)

# General Context

This lab’s focus is on what can be forensically found in a capture of memory from a running Windows machine. We utilize a program called Volatility (The Volatility Foundation - Open Source Memory Forensics, n.d.) that specializes in advanced memory analysis and forensics. When Volatility was introduced in 2007 it was the first time that it was shown how powerful it could be to analyze the runtime state of a system because of how much data is stored in volatile random access memory (RAM). Volatile memory means that it is not persistent, so as soon as the machine is shut down or rebooted the memory is wiped, so being able to capture what is in memory while it’s running for analysis can give a much clearer and deeper insight into what actually happened on the machine.

As shown in the previous lab, there is an incredible amount of information in the Windows Registry, but there is much more stored in RAM. Some examples of what is stored in memory but not persistently in the Registry are: browsing history, encryption keys, chat messages, clipboard contents, run-time system activity, open network connections, recently executed commands and processes, injected code fragments, and memory stored before shutdown or crash (Don’t Forget RAM Capture, n.d.). If an attacker infiltrated your network by going through a reverse shell that’s running on a machine, you can see, as shown in this lab, where the process was executed from what ports it was listening on and who started it. None of these options would otherwise be readily available with other forensic analysis. You would see the netcat executable, but you would have no idea if it was executed, with what parameters, nor if anyone connected to it.

All of this said, the only way to be able to analyze a memory capture is to capture it. There are many tools that allow this to be done easily, but arguably one of the most important steps of an incident response plan is planning phase. You need to have tools at the ready, and know how to use them because when the time comes that you need to capture memory, time is often very much of the essence.

# Technical Context

Volatility is another great example of an active open source community. The code itself is owned by the Volatility Foundation, but because it’s open source anyone and everyone can see how and what is running and contribute plugins to automate tasks that otherwise could be cumbersome, mundane, or something that just has to be executed in a very particular way. One of the quickest ways to make your job easier and look effortless is to automate the boring stuff.

We can look at two opposing examples from the lab, first we needed to find the DHCP lease end time for the machine, and secondly we needed to dump the usernames and password hashes. The DHCP settings and options are buried deep down in the registry. I couldn’t find any references for where it’s stored via google searches so I tried a few different approaches. My first attempt was to look through the ‘CurrentControlSet’ but that actually didn’t exist so I started browsing through ‘ControlSet001’ I spent some time looking through ‘ControlSet001\Services\Dhcp’ and it’s subkeys, before I realized to look in Tcpip instead of Dhcp. I did finally find the epoch timestamp that I was able to convert into an ISO timestamp for human readability, but it took me quite a long time to dig around and finally find what I was looking for. On the other side of things, I was able to call Volatility point it at the file and type ‘dumphashes’ and within a few seconds I had every account on the machine along with their password hashes, a few more seconds against a password cracker and I had the two unique passwords in plain text.

If I thought that the DHCP information would be consistently useful, I could easily write a plugin that automatically pulled the TCP/IP Interface settings and could even convert the timestamp automatically. It would take my hour or two of fumbling and be able to get the same output in a few seconds. When dealing with the Windows Registry and Windows Memory contents it’s amazing how much data you can find, but you need to use the tools available to you in order to make your time searching worthwhile.r

# Solution

We start this lab by logging in to the ‘Introduction to Forensics’ lab environment in the US Cyber Range. We then switch directories to ‘~/Desktop/cases/02\_Memory-Image/’ and run `vol.py -f WindowsVista\_0401.vmem imageinfo` to run Volatility against the image and output some basic information about the image. We learn that it’s likely from a machine running Windows Vista Service Pack 2, and the image was taken on 2016/04/01 at 21:09:25 UTC.

Graphical user interface, text

Description automatically generated

Figure - changing working directory to where the image is

Text

Description automatically generated

Figure - printing the Volatility help menu

Text

Description automatically generated

Figure - imageinfo tells us about the image file

Our next steps are to do some basic Volatility commands to see what processes were running, what network connections were open, and what Users are on the machine. Running ‘pslist’ runs the same query that ‘tasklist’ would run if Windows was running, where ‘psscan’ doesn’t trust what is in the list and tries to scan through the memory to identify it’s own list of running processes (Difference between Pslist and Psscan, n.d.) this can be very helpful in identifying hidden processes.

Text

Description automatically generated with low confidence

Figure - listing the running processes with pslist

Graphical user interface, text

Description automatically generated

Figure - listing the running processes as found by psscan

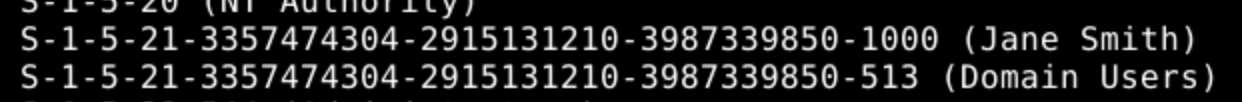


Figure - running getsids to identify the accounts

Graphical user interface, text

Description automatically generated with medium confidence

Figure - running getsids and grepping for all of the processes running as a specific user

Text

Description automatically generated

Figure - running getsids and searching for a specific executable to see all of the SIDs that own it

Next, we can search specific Registry keys if we are looking for a specific entry like the specific build number of the operating system service pack, or the timestamp that the OS was installed. We can even list the services that are actively listening for network connections, the ports and IP that they are listening on, and even if any outside IP was currently connected to this machine. For example, we can see lsass.exe was listening on port 49155.

Text

Description automatically generated

Figure - registry key showing the OS version and build number

Text

Description automatically generated

Figure - registry keys showing the version, service pack, and build number

A picture containing text

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Figure - the timestamp in the registry and it converted to human readable format

Table

Description automatically generated

Figure - a listing of the processes listening for network connections

Using ‘cmdscan’ we can see all of the different commands that were run at the command line. From this, we can tell that netcat was executed to listen on port 23 and give a reverse shell command prompt to anyone who connected in to it, but that instance was process ID 532, and no longer running while it was currently running as PID 832. We then view the process handles for PID 832, and search for shellbags which can provide a wealth of information like metadata, recently used files, access and create timestamps (Levy, 2012).

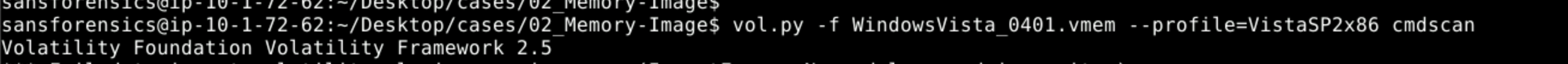


Figure - executing the cmdscan plugin

Text

Description automatically generated

Figure - PID 532 was a netcat reverse shell listening on port 23

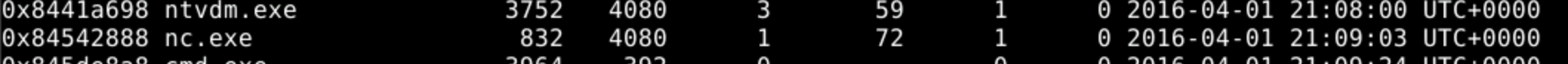


Figure - netcat in the process list

Text

Description automatically generated

Figure - displaying the process handles for netcat running on PID 832

Graphical user interface, text

Description automatically generated

Figure - running the shellbags plugin to find more information about netcat

We still want to know more information about who and what ran netcat. Looking closer at ‘cmdscan’ the Jane Smith user entered C:\Users\Jane Smith\ Desktop\netcat-1.11\ and listed the directory contents, edited the readme file and then started it as a reverse shell on port 23. Then running the ‘envars’ for PID 832 shows us the environment variables associated with the running process like what APPDATA path was associated with it, and what it used for a temp directory.

Text

Description automatically generated

Figure - showing the commands typed in before netcat was executed

Text

Description automatically generated

Figure - showing environment variables for PID 832

Finally, the lab has us find out two last pieces of information. It has us identify the DHCP information and when the lease was set to end for the assigned IP address, and then to dump the user hashes. The two passwords associated with the accounts were ‘password’ and ‘p@ssword’ and both were cracked in a matter of seconds.

Text

Description automatically generated

Figure - printing the DHCP information

Graphical user interface, text, application

Description automatically generated

Figure - converting the epoch timestamp to human readable format

Graphical user interface, text

Description automatically generated

Figure - dumping the user information and password hashes

Shape, rectangle

Description automatically generated

Figure - cracking the password hashes

# ****References****

*Difference between pslist and psscan*. (n.d.). Retrieved November 24, 2021, from <http://akovid.blogspot.com/2014/02/difference-between-pslist-and-psscan.html>

*Don’t Forget RAM Capture: A Key to Digital Forensics | 60 second video*. (n.d.). Retrieved November 24, 2021, from <https://www.adfsolutions.com/news/ram-capture>

Levy, J. (2012, September 25). Volatility Labs: MoVP 3.2 Shellbags in Memory, SetRegTime, and TrueCrypt Volumes. *Volatility Labs*. <https://volatility-labs.blogspot.com/2012/09/movp-32-shellbags-in-memory-setregtime.html>

*The Volatility Foundation—Open Source Memory Forensics*. (n.d.). Volatilityfoundation. Retrieved November 24, 2021, from <https://www.volatilityfoundation.org>