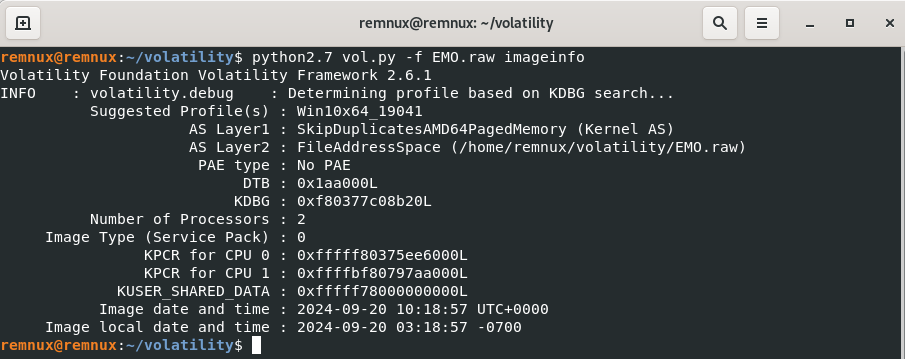
In this study, we utilized plugins from both Volatility 2 and Volatility 3, as certain plugins were not available in one version or the other. This approach allowed us to leverage the strengths of each version to conduct a comprehensive memory forensics analysis

(Volatility 2 - 2.6.1 framework And Volatility 3- 2.7.1 framework)

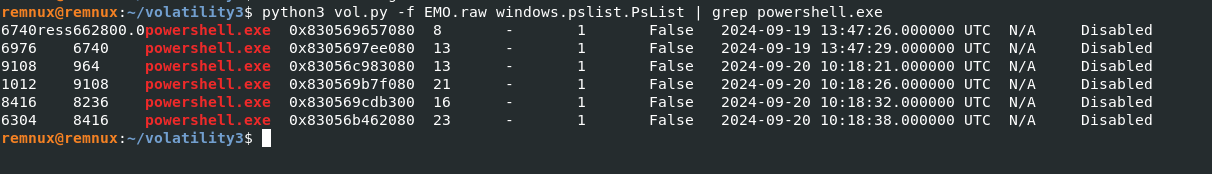
Using imageinfo plugin we need to determine the profile for volatility2 version as we need to specify the profile.



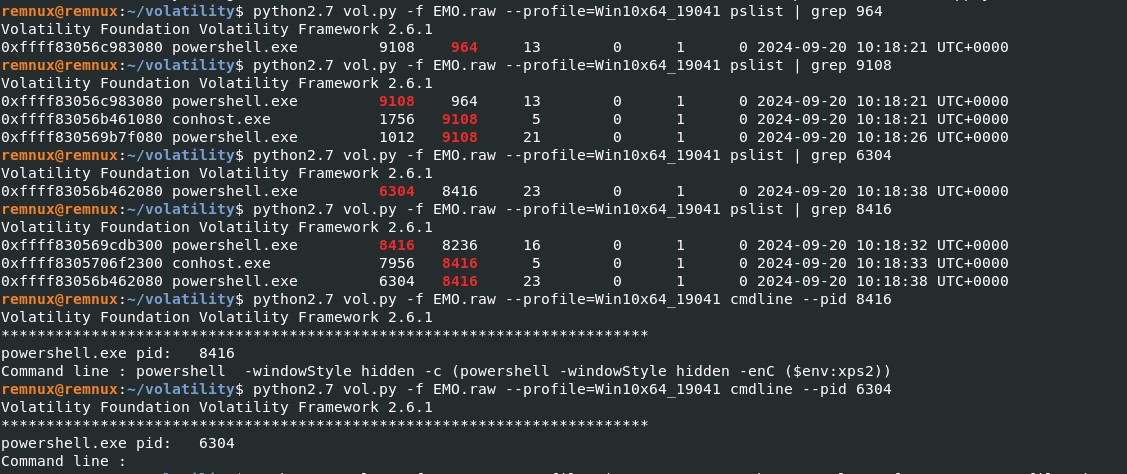
From this plugin’s output we get to know that the Profile of the memory image is **Win10x64\_19041**

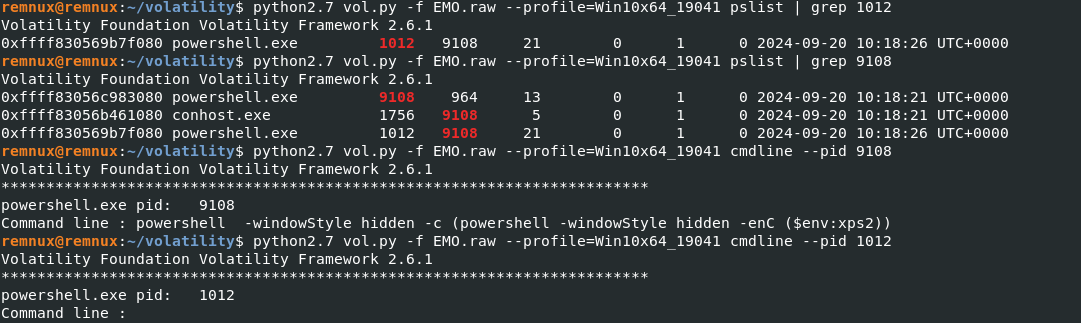
We used psscan and psList plugins to determine the process ID’S , Parent and Child process, we found multiple instances of powershell process and We noticed several PowerShell processes, which seemed suspicious, so we decided to track them further

Volatility 3

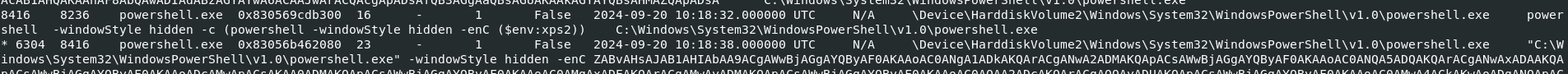


Volatility 2

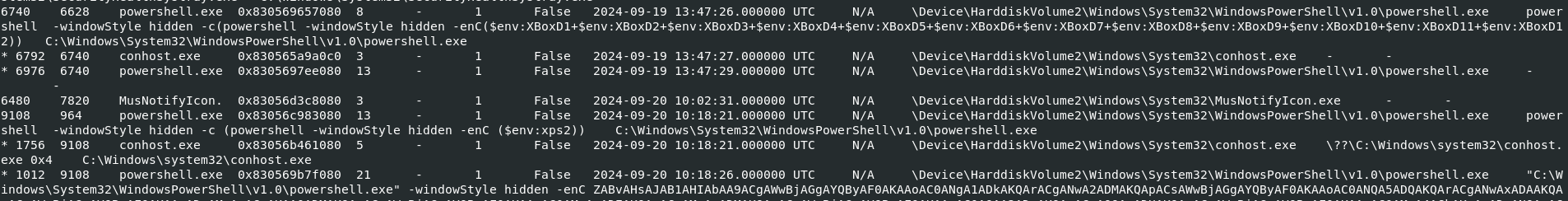


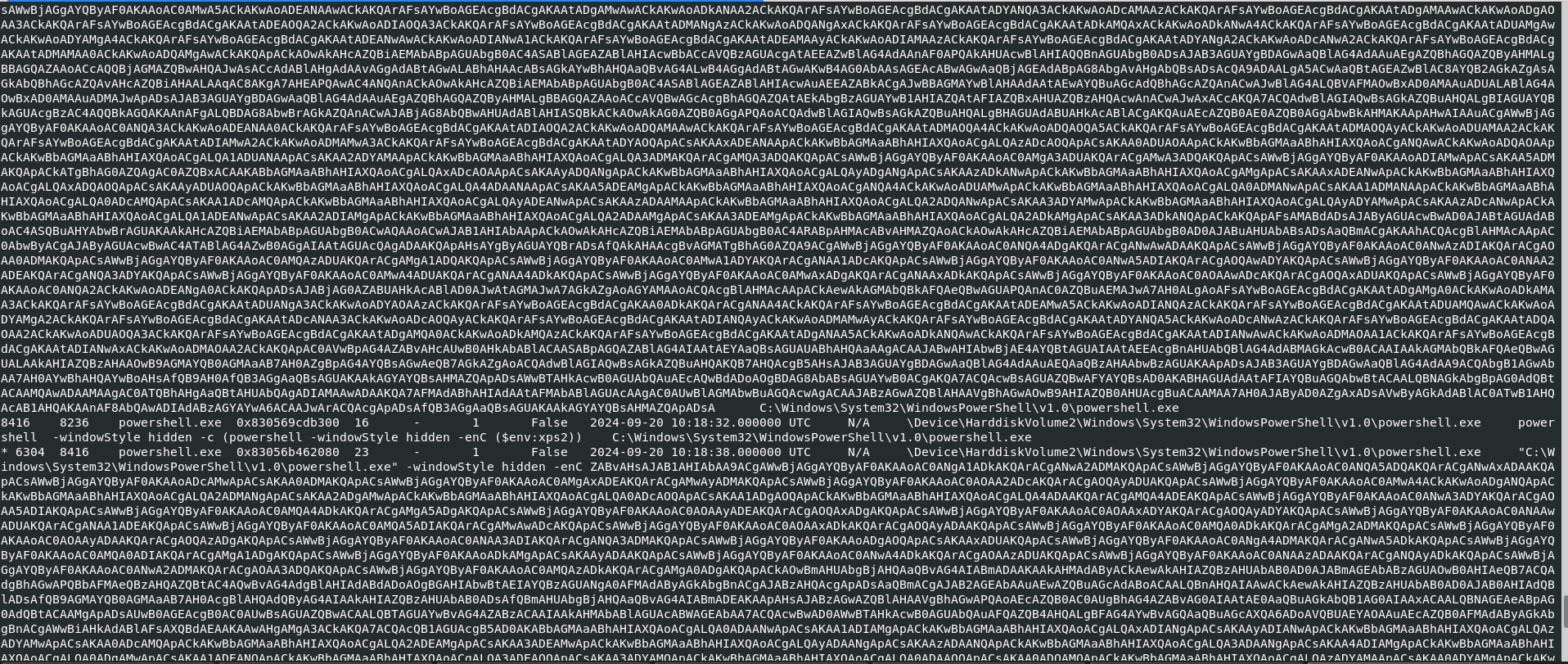


PowerShell processes exhibiting hidden activity. The process IDs of these suspicious processes were noted for further analysis:









**Multiple PowerShell Processes**: Several instances of powershell.exe are being executed from the C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe location. Each process is running with the -windowStyle hidden flag, meaning the PowerShell windows are not visible to the user.

The PowerShell commands being executed are base64 encoded

**Persistence or Remote Execution Indicators**: The commands could be associated with persistence mechanisms or remote code execution. Using PowerShell to run hidden processes with encoded commands is a common technique in malware and fileless attacks,

When a PowerShell script or command is executed, especially if it's interacting with the console, a conhost.exe process is often spawned to handle the console window. This is part of normal system behavior. However, in your case, the PowerShell instances are **hidden** (-windowStyle hidden), meaning the console windows are suppressed from view, but the processes still exist in the background.

* When PowerShell runs commands in hidden mode, it still needs conhost.exe to manage the execution of commands behind the scenes, even if the user doesn't see it.
* Normally, when you open a PowerShell window manually, you'll see both the powershell.exe process and the associated conhost.exe process. But, if an attacker is trying to hide their activity, they’ll suppress the PowerShell window with the -windowStyle hidden flag, so you won't see the windows on the desktop, but the processes are still running in the background.

**Persistence**: The persistence of both PowerShell and conhost.exe processes over a period of time, combined with their hidden execution, suggests that the system might be compromised.

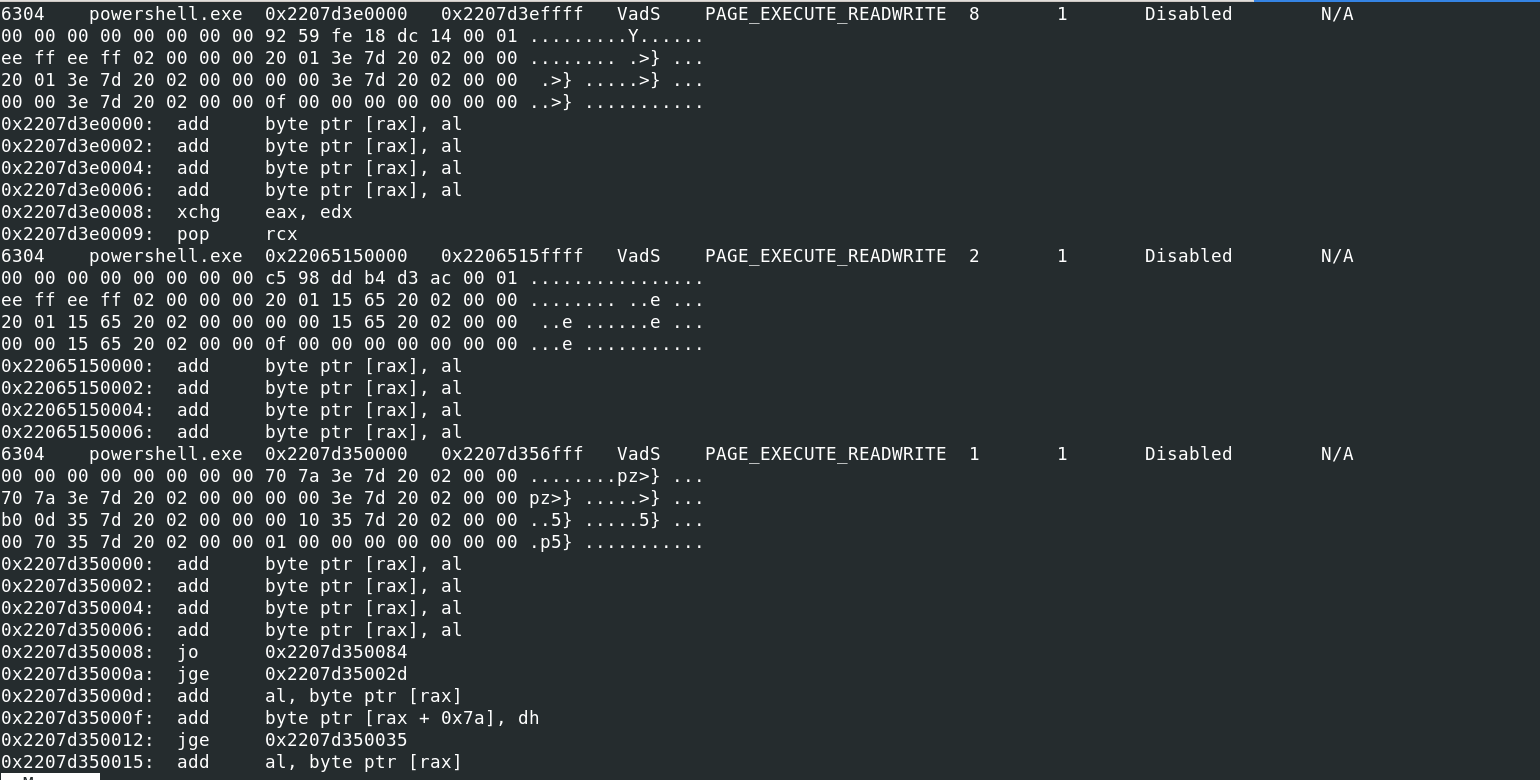
With the help of the output from psscan we observed that the obfuscated code /command is executed in powershell with the process id **6304**

**MALFIND**

Using malfind we also spotted this process , on top of that we are examining the memory region   
Of this process and try to understand what is happening in that memory region

The memory regions with PAGE\_EXECUTE\_READWRITE permissions, especially in a PowerShell process, could indicate suspicious or malicious activity. This is because:

* The ability to both write and execute code in memory is often associated with malicious code injection techniques (e.g., shellcode injection, reflective DLL injection).
* The pop rcx and jmp rax instructions are often found in exploitation or attack payloads.

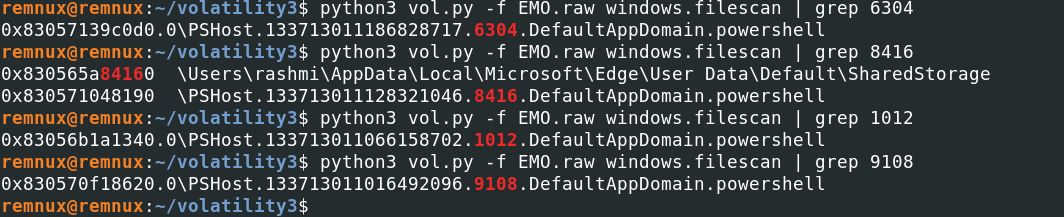


The powershell process with pid (6304,8416,1012,9108) obtained common from malfind psscan/pslist is examined for dropping of files on harddisk, using filescan

With this we came to know that there were no executables file associated with these suspicious process

**Calls to arbitrary memory locations (like jmp rax)** suggest that dynamic code is being executed, which is a common tactic in exploitation (e.g., executing shellcode from an arbitrary memory location)

Volatility 3

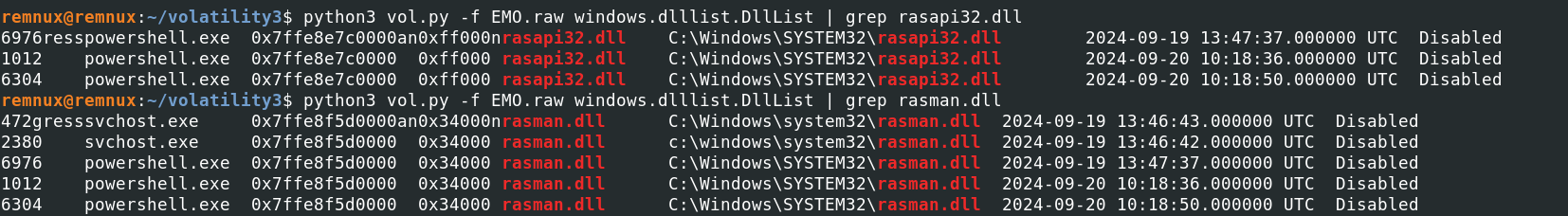


Using the filescan plugin, we were unable to locate any executable files associated with the suspicious processes on the hard disk. This indicated that the malicious process did not rely on or drop any executable files. Instead, it operated entirely in memory, utilizing legitimate system tools to execute its operations. This behavior suggests that the malware exhibited fileless characteristics. To confirm its malicious nature, we proceeded with additional plugins to gather more evidence.

**Dlls used by suspicious process**

**rasapi32.dll and rasman.dll**: These are related to the Remote Access Service, which could be used for remote access connections. This could suggest the involvement of remote management or access.

Volatility 3

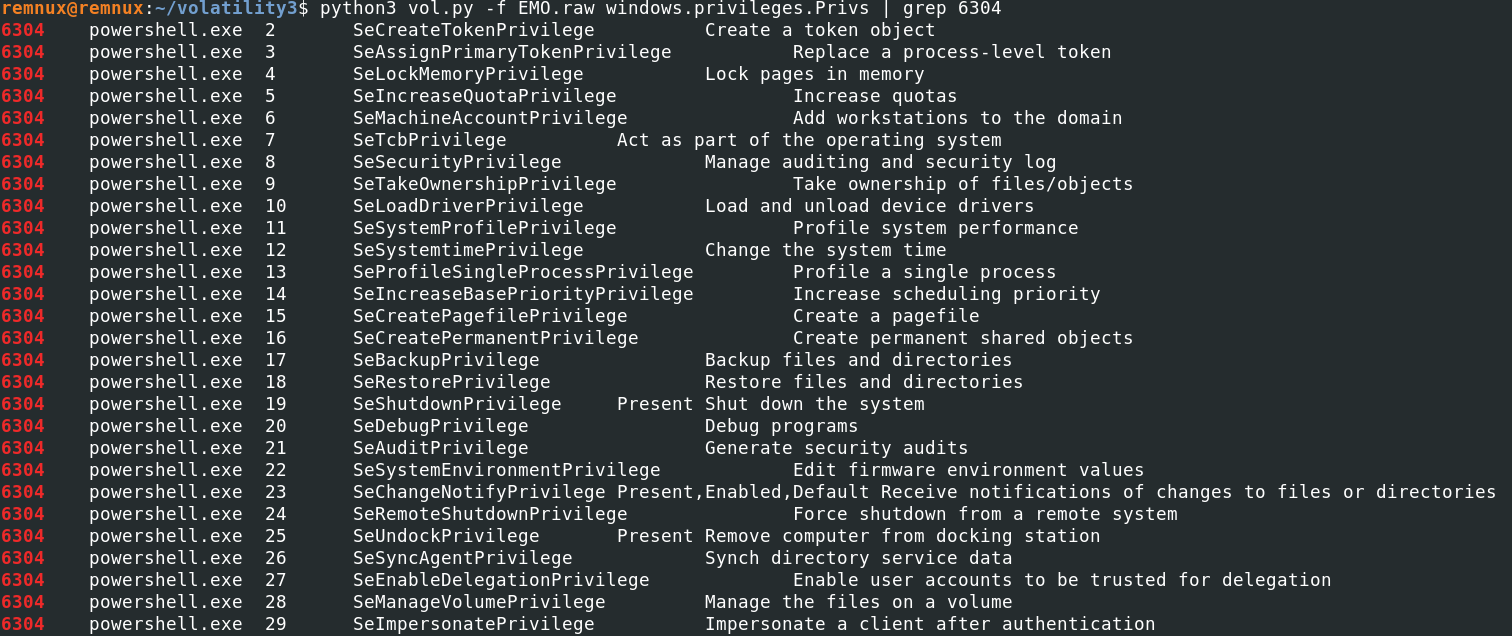


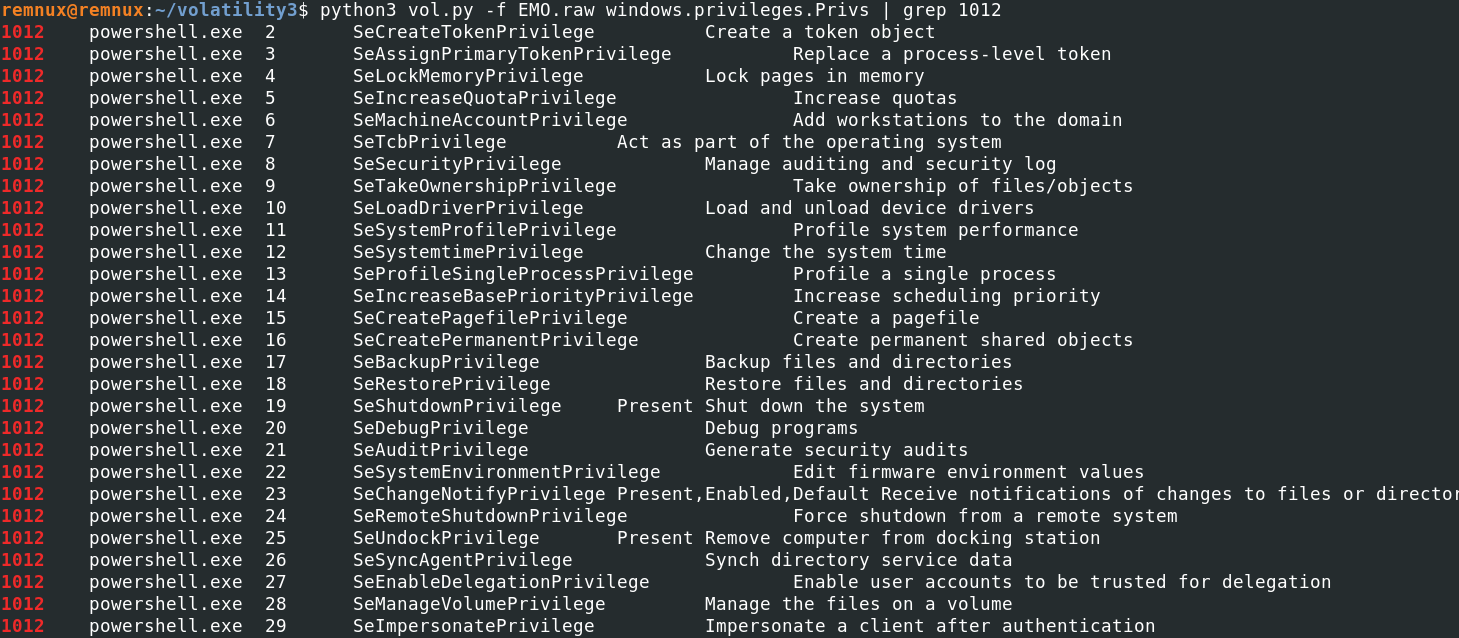
As we can see these dlls are used by the suspicious process (6304,1012)

All the dlls used by these process are legitimate but it can be leveraged to perform malicious activities.

**Privileges of Suspicious Process:**

Volatility 3



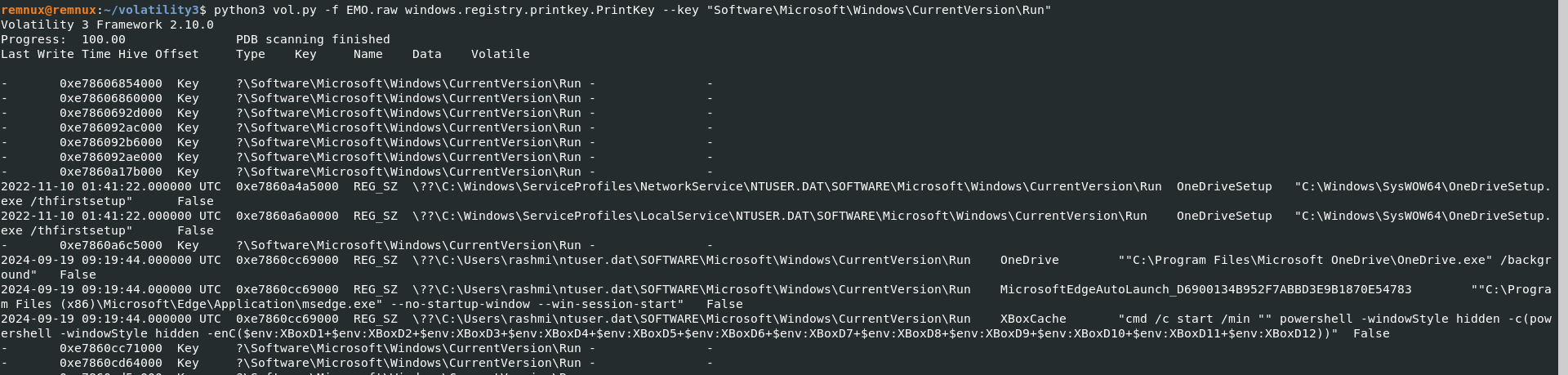


**Both processes (PID 6304 and PID 1012) represent potentially high-risk instances of powershell.exe** with extensive system privileges.

**HANDLES AND PrintKey**

We further utilized the handles plugin to inspect the handles associated with the suspicious processes, which provided insights into the files, registry keys, and other system resources these processes interacted with. Additionally, the printkey plugin was employed to examine specific registry keys, helping us track any registry modifications made by the malicious processes

Volatility 3



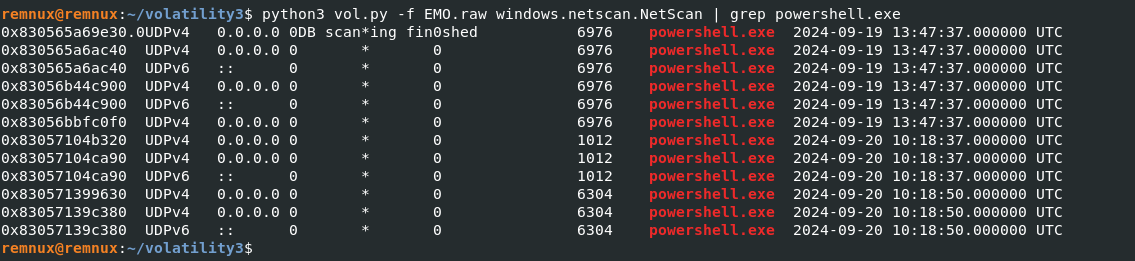
The Run key entry executes a hidden PowerShell command on system startup. It uses the cmd /c start /min command to minimize the window and execute the command in the background (powershell -windowStyle hidden).

The PowerShell command concatenates multiple environment variables ($env:XBoxD1 to $env:XBoxD12) and executes the resulting payload using -enC. The -enC flag indicates that the PowerShell script is encoded in base64.

The concatenation of environment variables and the use of -enC for encoding the command are commonly used techniques in obfuscated or malicious scripts.

This output further supports the hypothesis of fileless malware using PowerShell for persistence or execution.

Volatility 3



The local IP address for many connections is 0.0.0.0, which suggests that the process was listening on all network interfaces for incoming connections.

No external IP addresses are visible, indicating that these might be local network connections or unresolved connections.

powershell.exe is typically a system administration tool, but in this case, it has established multiple network connections. PowerShell is commonly abused by attackers for remote administration and running scripts (often referred to as "living off the land" tactics). The fact that multiple powershell.exe processes are communicating over the network, especially with UDP (which is connectionless and harder to track), could indicate malicious usage.

There are two distinct time ranges for these network connections:

* **September 19, 2024, 13:47:37 UTC**: Several UDP connections associated with powershell.exe (PID 6976) were active during this time.
* **September 20, 2024, 10:18:37 - 10:18:50 UTC**: A different set of powershell.exe instances (PIDs 1012 and 6304) established new connections on UDP ports 1012 and 6304.

The first group of connections on **September 19** (PID 6976) and the second group on **September 20** (PIDs 1012 and 6304) suggest that these PowerShell processes were not short-lived but persistent or repeatedly launched at different times. This indicates either scheduled tasks, recurring scripts, or persistent backdoors that are maintaining or establishing communication channels.

In this case as we encountered multiple instances of powershell in earlier results we can say that is due to recurring scripts

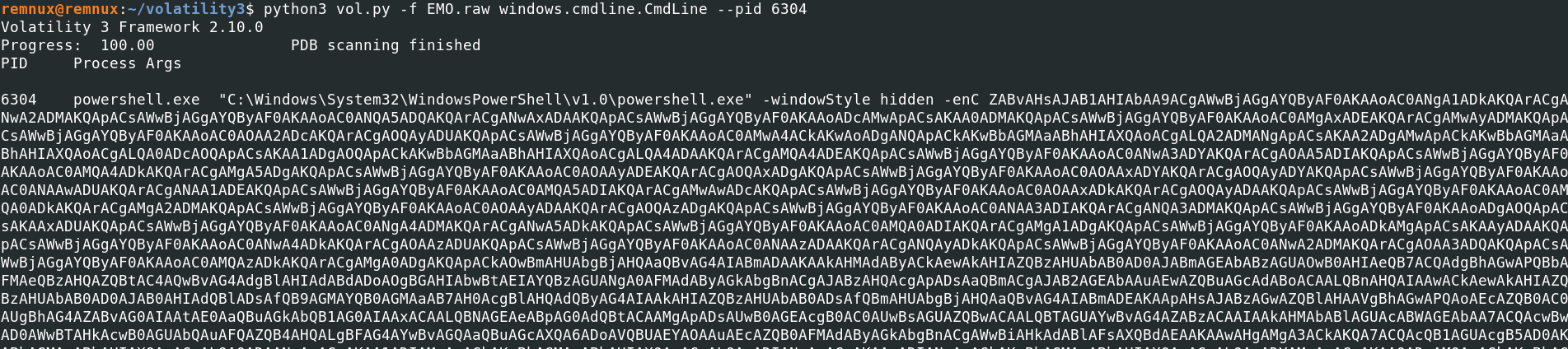
We were not able to track the complete network activity but we tried alternative ways to gather more evidence.

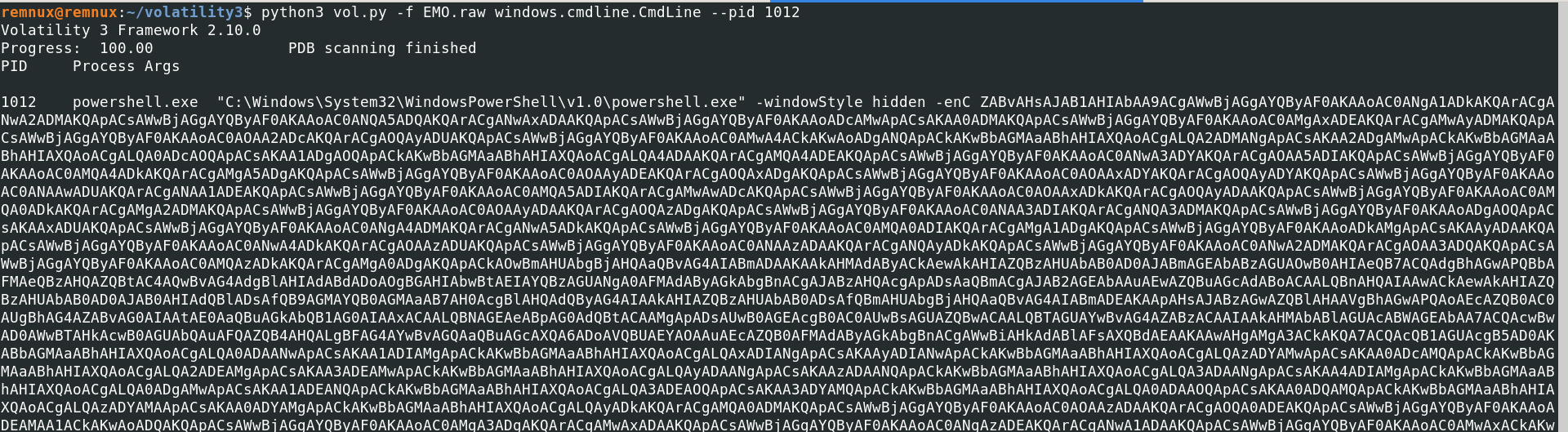
**Command line activity**

Since we didn't find any executable files, we can conclude that the attack is of type script based file less attack, also we can make use of plugins of volatility to further dig deeply to track what was the code trying to do using cmdline.CmdLine plugin.

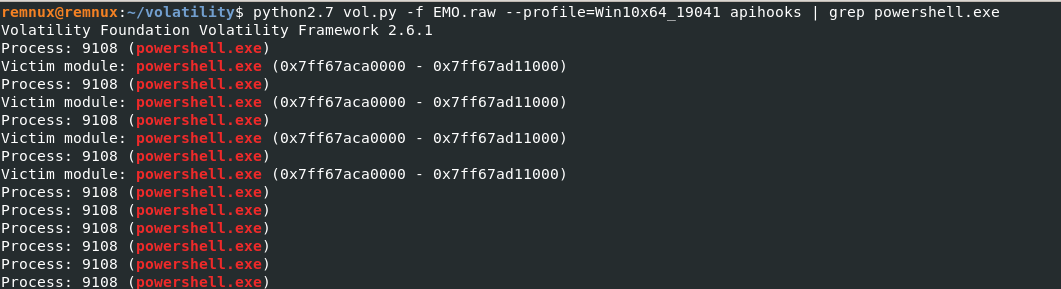
The results of which are described below.

Volatility 3



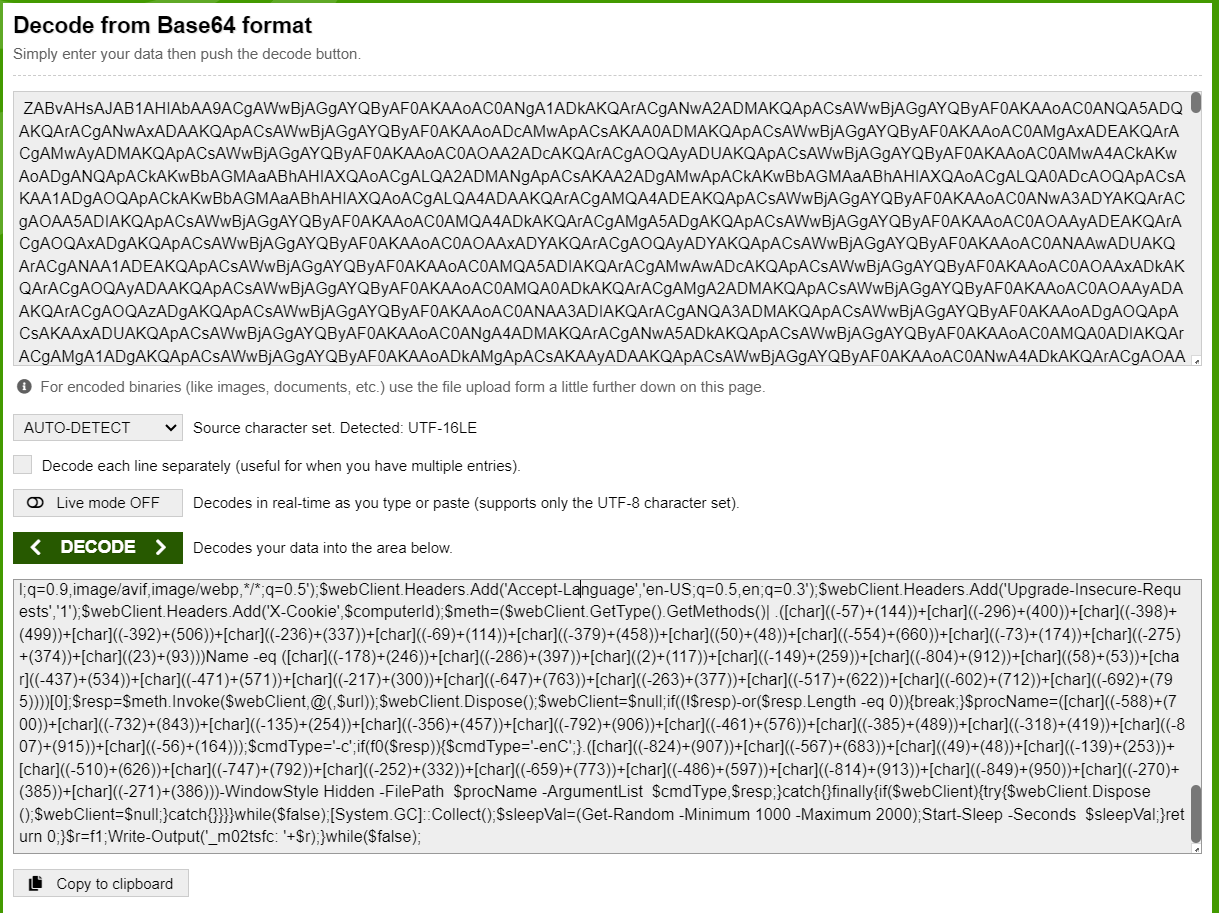


Using cmdline plugin , we can get the commands executed , we can see that both processes(6304,1012) executes highly obfuscated commands.



specifically filtering for powershell.exe processes. It indicates that powershell.exe (Process ID 9108) has several API hooks, which is a common technique used by malware to intercept or alter system API calls for malicious purposes. The hooks observed here could suggest that the PowerShell process has been compromised to inject or manipulate system calls

**COMMANDS**



Link to the script:

Link to the decoded script:

Major observations from one step decoding:

### **Script Structure and Key Components:**

1. **URL Construction Using** [char] **Expressions**:
   * The script builds a URL dynamically using a series of [char] expressions, such as [char]((-659)+(763)). This technique hides the actual URL, which can be decoded by evaluating these expressions.
2. **Functions f0 and f1:**
   * The script defines functions to check if strings are Base64-encoded (f0) and to introduce random delays and sleep intervals (f1), likely to evade detection or analysis.
   * It constructs a query, gathers system information (e.g., serial number, computer name), and hashes them using SHA-256 and MD5 to generate a unique identifier for the infected machine.
3. **Web Requests Using** WebClient**:**
   * The script generates a custom User-Agent and sends HTTP requests using WebClient. It also includes headers like Accept and X-Cookie, potentially used for exfiltrating system information or communicating with a remote server.
4. **Execution of Encoded Payloads:**
   * It may retrieve a response, check if it is Base64-encoded, and execute it with different PowerShell flags (-c or -enC).
5. **Persistence and Looping:**
   * The script runs in an infinite loop (while($true)), introducing sleep intervals and repeating the communication process indefinitely.

**Since the code is highly obfuscated, we were able to infer only this much of information**

**If u want more u can analyze with other plugins.**