Report

Title: Comparative Memory Forensics: Analyzing Volatile Data Before and After Internet Activity

Riphah International University Islamabad

Name	Sap Id
Saad Naveed	43973
Muhammad Masab Qayyum	46472
Course	Digital Forensics
Course Instructor	Mr. Humayun Raza
Date	5th May 2025

Contents

Introduction:

Memory forensics is used to see what was happening in a computer's RAM. It helps us find out which programs were running, which websites were visited, and what network connections were active.

In this project, my teacher gave me the task to do memory forensics on **Windows 11**. I downloaded the Windows 11 ISO file and installed it on a **virtual machine (VM)** to do the testing.

I checked the memory in two cases. First, I took a memory dump when the system was clean and not connected to the internet. Then, I connected it to the internet, opened a few websites, and took another memory dump.

I used **FTK Imager** to take the memory dumps and **Volatility 3** to check and analyze them. These tools helped me see what changed in memory before and after using the internet.

This project helped me understand how internet activity changes the data in memory and why memory forensics is useful for finding user activity and digital evidence.

Objective

The objective of this project was to analyze two memory dumps taken from a Windows 11 system. The first dump was taken when the system was clean and not connected to the internet, while the second dump was taken after connecting to the internet and visiting websites. The goal was to examine both memory dumps and find out what forensic artifacts were present in each dump and how internet usage affected the data stored in memory.

Tools Used

In this project, I used two main tools to work with memory forensics:

FTK Imager:

FTK Imager is a **GUI-based** (Graphical User Interface) tool. It's easy to use and helps you take memory dumps from the system. The tool lets you create an image of the system's memory (RAM) with just a few clicks. It supports different file formats and is perfect for capturing data from live systems for forensic analysis.

Volatility 3:

Volatility 3 is a **CLI-based** (Command-Line Interface) tool. This one is a bit more advanced, and you have to use commands to run it. It's powerful for analyzing memory dumps and finding important forensic information like running programs, network connections, and other activities. Since it's command-line based, it's better for those who are familiar with using commands in a terminal.

Methodology

1. Download Windows 11 ISO
   First, I downloaded the Windows 11 ISO file from the official Microsoft website.

2. Set Up Virtual Machine (VM)

   Then, I made the ISO bootable using a USB and installed Windows 11 on a virtual machine (VM).

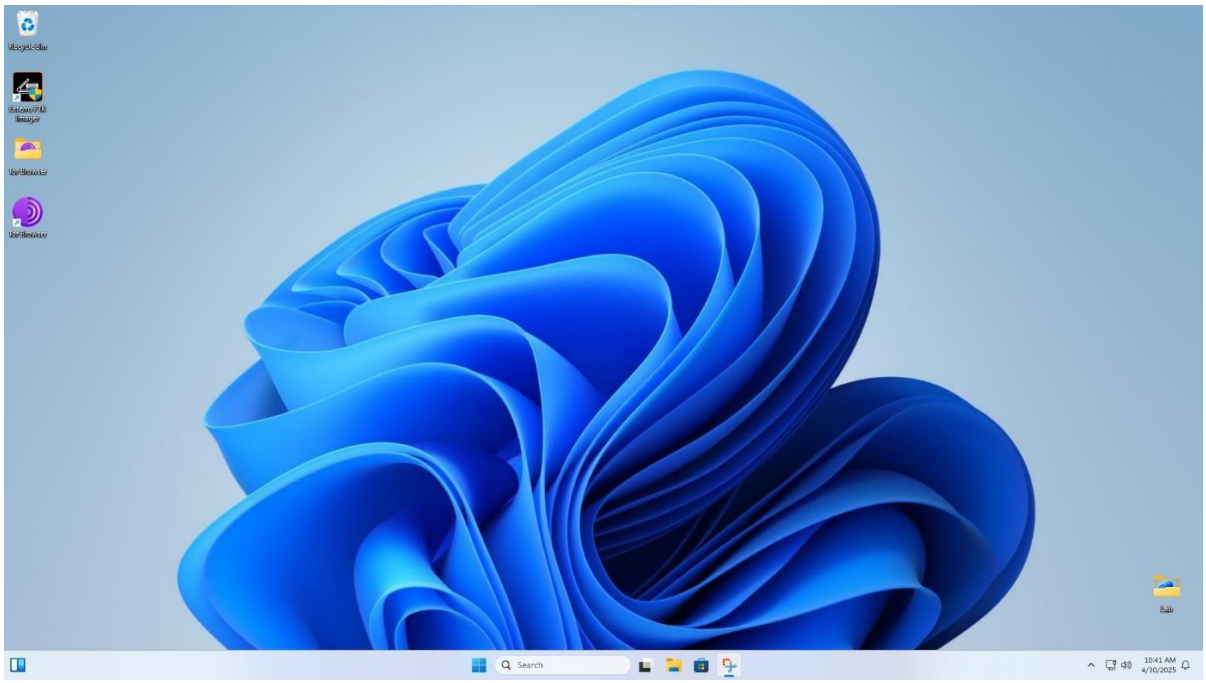

Figure 1 Install Window 11 in VM

3. Install Tools

   I copied **FTK Imager** and **Volatility 3** from my host system to a USB. Then, I went into the VM and installed both tools there.

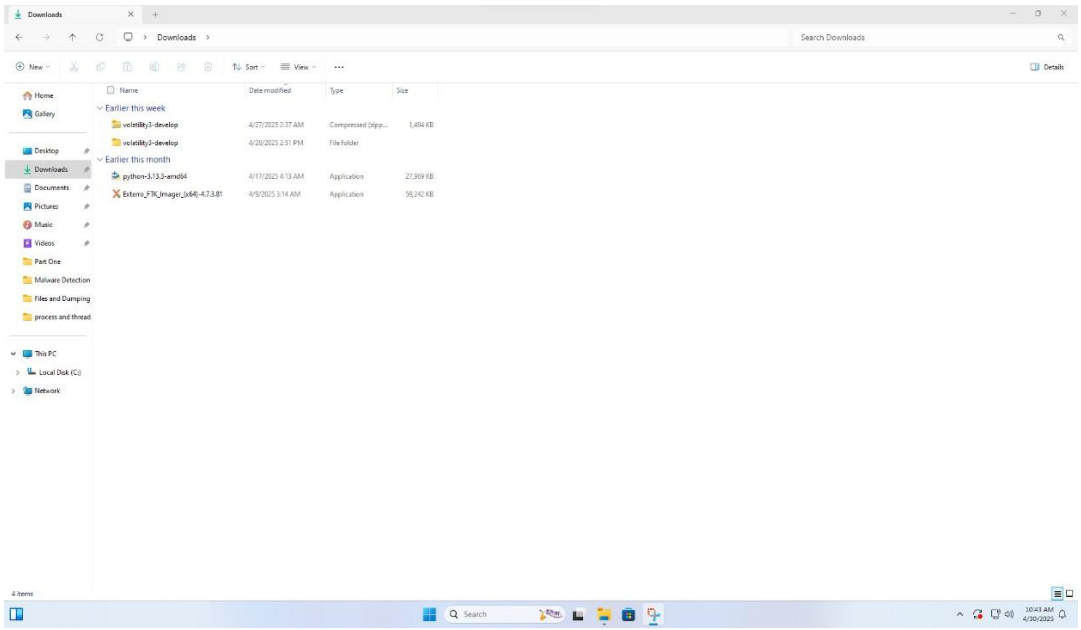

Figure 2 Installs Tools

4. Take First Memory Dump (Before Internet)

I opened FTK Imager and took the first memory dump when the system was not connected to the internet.

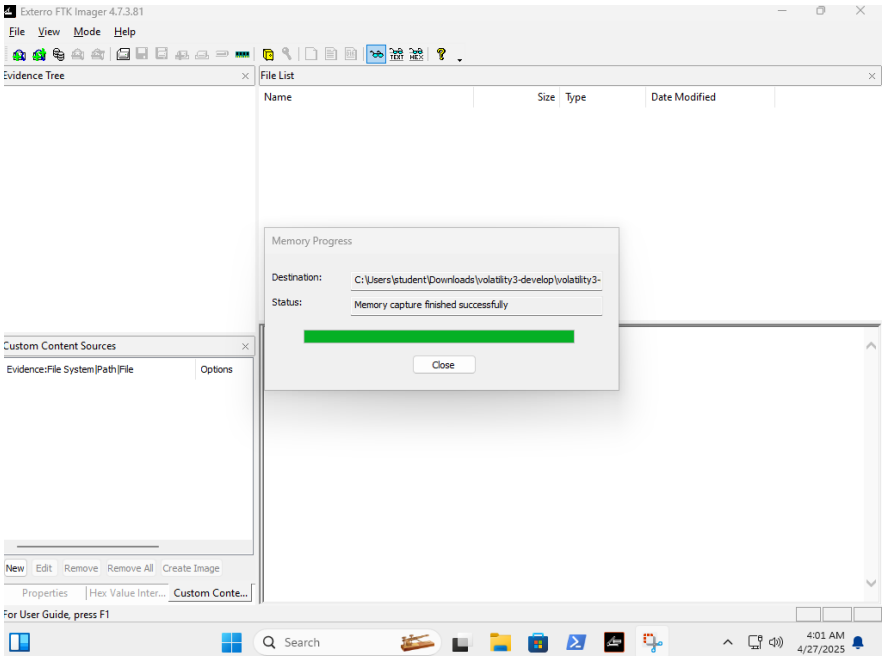

Figure 3 Clean Memory Dump

5. Use Internet

   After the first dump, I connected the VM to the internet and visited a few websites.

6. Take Second Memory Dump (After Internet)

   Then, I took the second memory dump after using the internet.

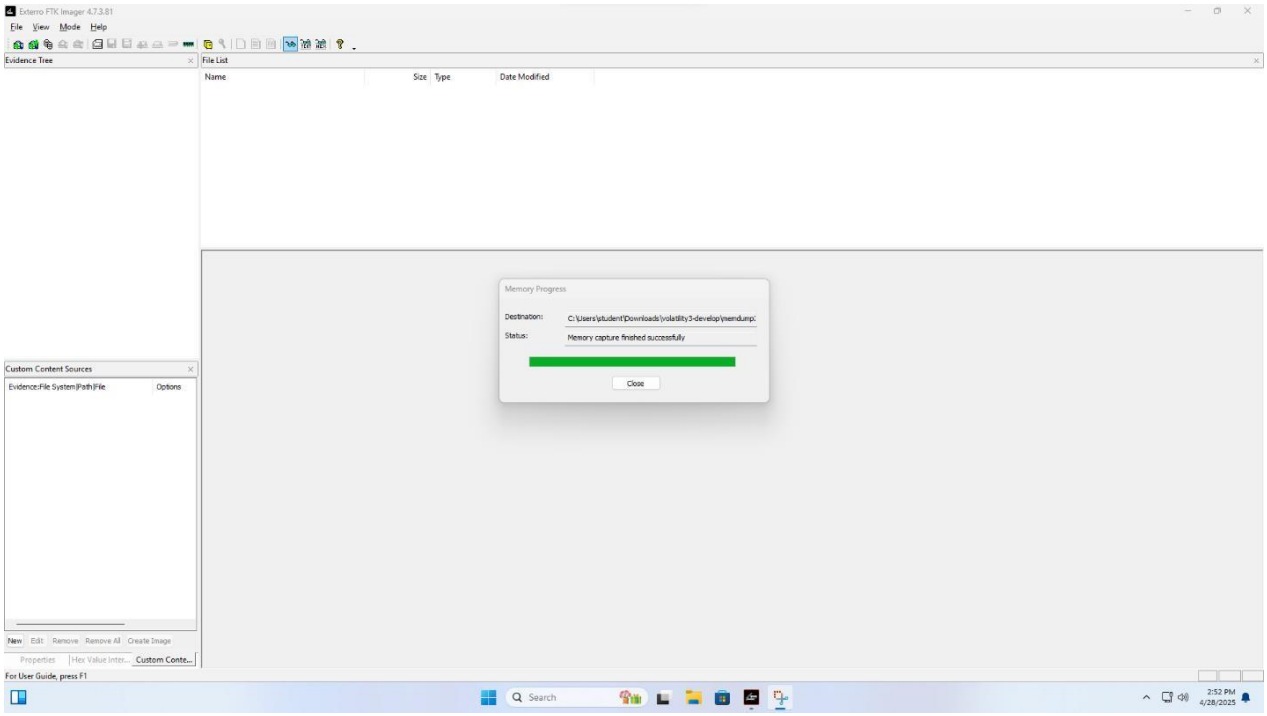

Figure 4 Second Memory Dump

7. Memory Dump Size

   Both memory dumps were around **5 GB** in size. I saved them inside the Volatility folder.

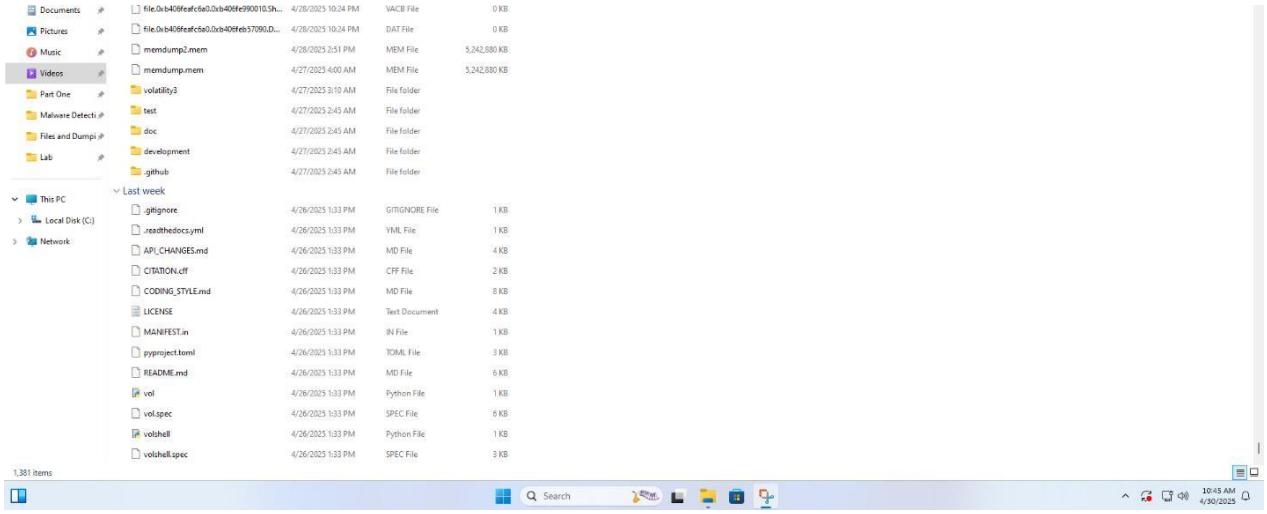

Figure 5 Save Both dump Volatility Folder

8. Install Python
   Since Volatility 3 needs Python, I copied the Python setup from my host system to the VM and installed it in Windows 11.

9. Analyze the Memory Dumps
   Finally, I used Volatility 3 to analyze both dumps and see what changed in memory before and after using the internet.

Phase 1

In the first phase, we took a clean memory dump. We opened FTK Imager and started capturing the image.

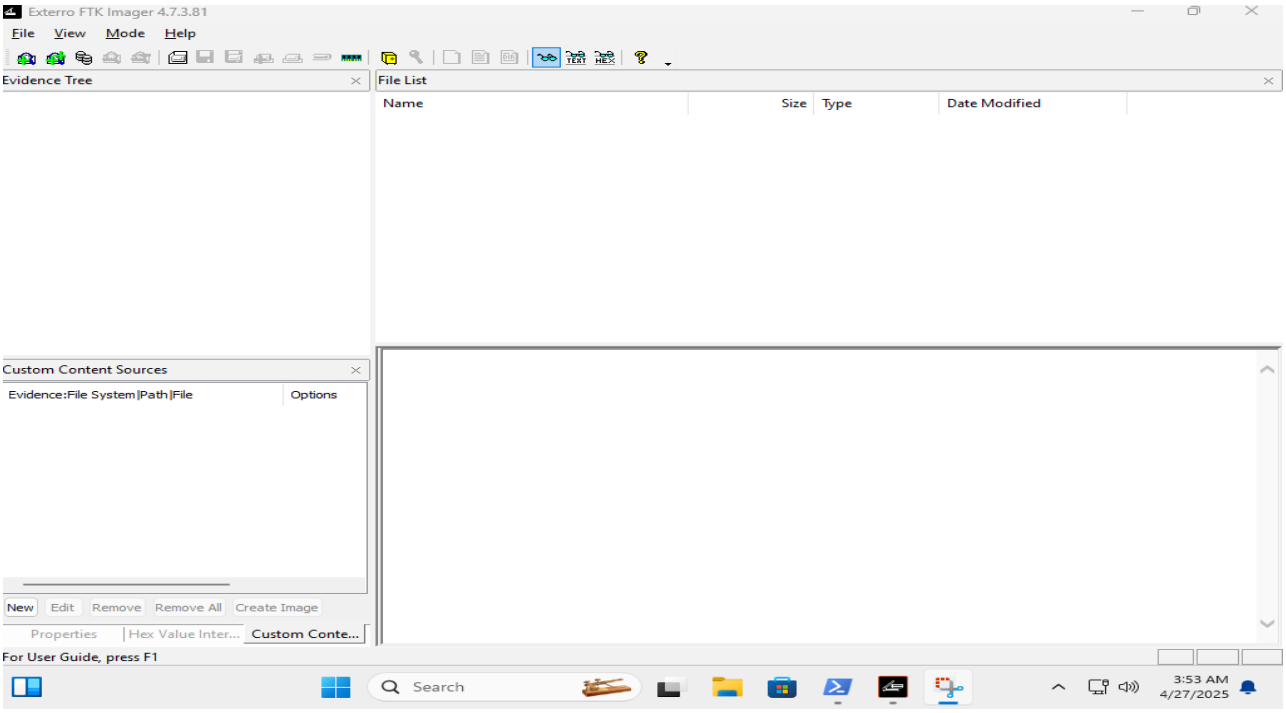

Figure c Open FTK

Start Capturing memory and select folder where save this file.

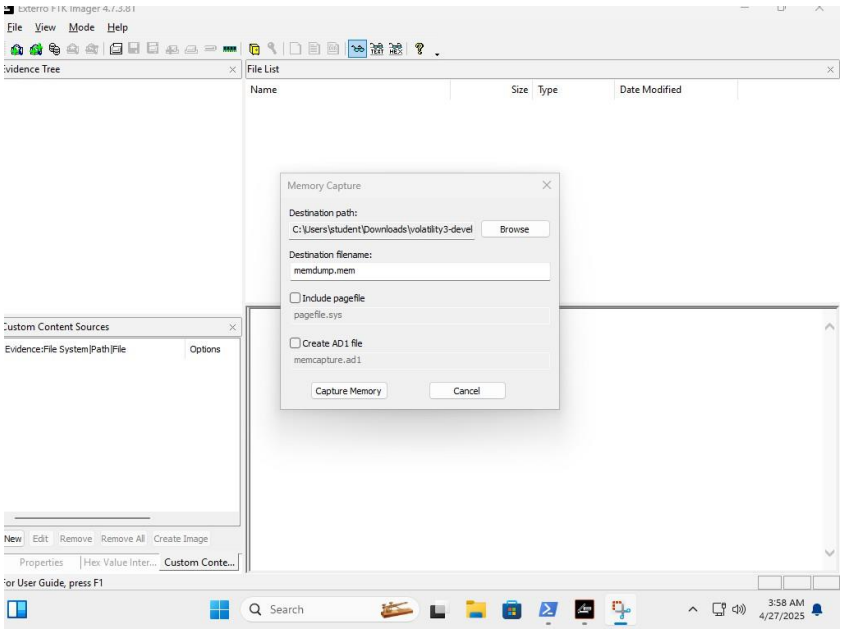

Figure 7 Save Dump file

Finaly, I capture clean memory dump.

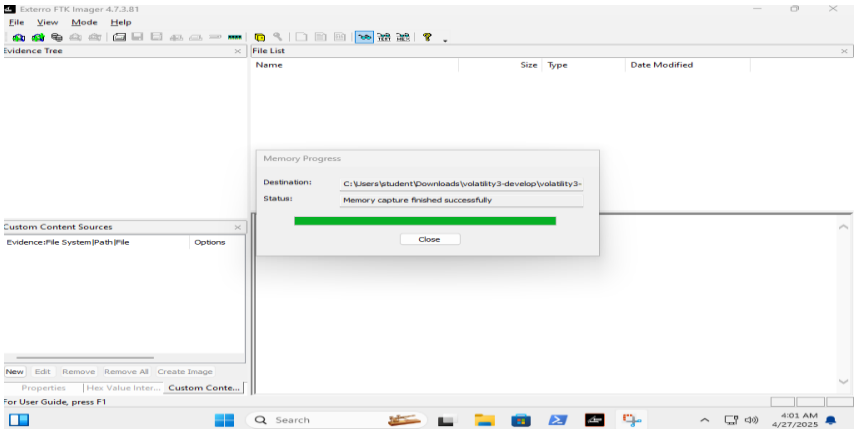

Figure 8 Successful Memory Capture

Analysis Phase 1

I run Volatility 3 frame work.

The pslist plugin shows all the **active and recently terminated processes** found in memory — these are important to understand **what programs were running**, their **process IDs**, **parent-child relationships**, and how the system was being used.

I run this command: python.exe .\vol.py -f .\memdump.mem windows.pslist

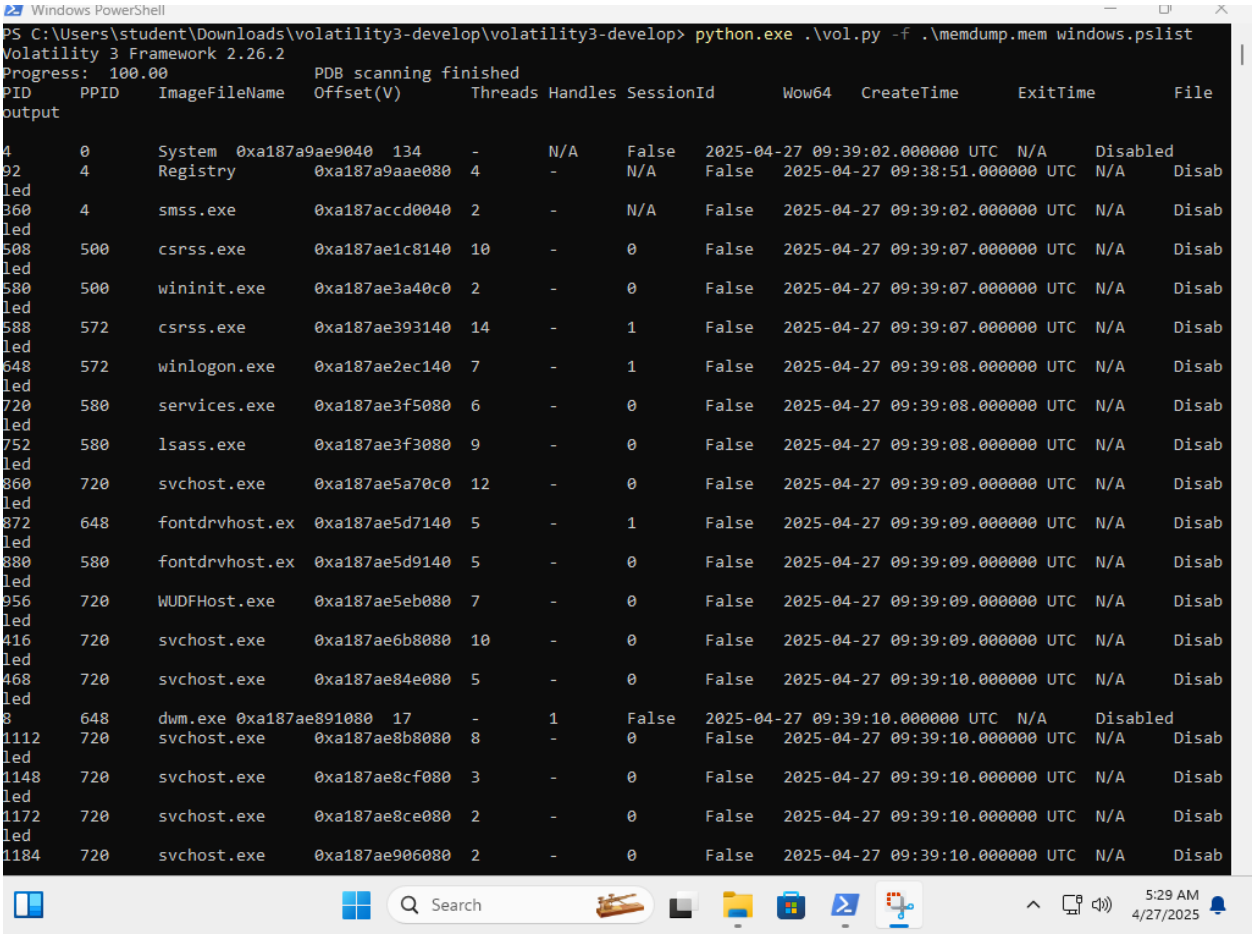

Figure S Volatility pslist Plugin

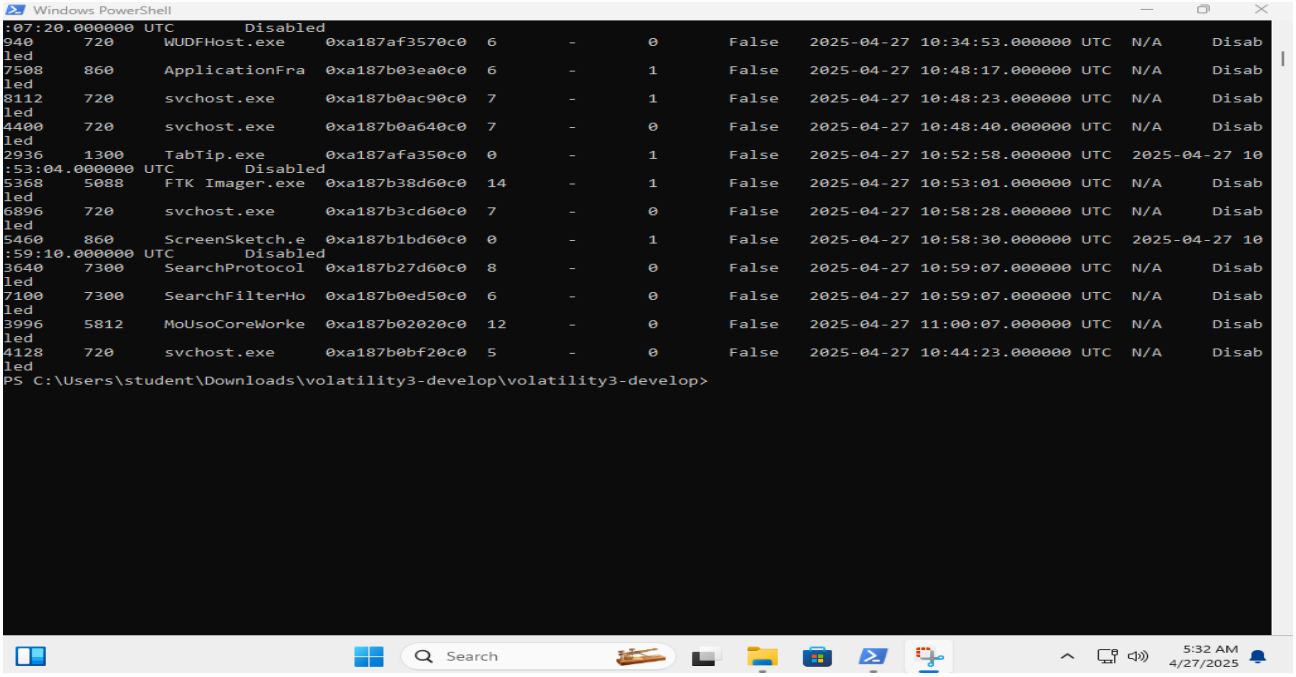

Figure S.1 Volatility pslist Plugin

Table 1Authentic Forensic Artifacts Identified:

Artifact	Details from Screenshot
Tool Used	Volatility 3 Framework 2.26.2
Command Used	python.exe .\vol.py -f .\memdump.mem windows.pslist
Processes Detected	Includes standard Windows processes like System, Registry, csrss.exe, wininit.exe, etc.
Timestamps	All processes created on 2025-04-27 around 09:39:00 UTC, suggesting a system boot or snapshot.
Suspicious Activity	No obviously suspicious processes are seen yet — all appear to be legitimate Windows services.

Interpretation Notes

We can see that there are **many svchost.exe processes**, which is **normal** because Windows runs different services using this file.

There are also important system processes like:

- csrss.exe

- winlogon.exe
- services.exe
- lsass.exe
- smss.exe

These show that the memory dump is from a **clean and working system**, probably taken **right after the computer was started**.

We don't see any **unknown or suspicious programs**, like strange .exe files or hacking tools. So, everything looks **safe and normal** in this phase.

The windows.info Plugin

The windows.info plugin shows **basic system information** from the memory image — this includes details about the **Windows version**, **build number**, **system root**, and other important configuration data.

I run this volatility plugin: python.exe -f .\memdump.mem windows.info

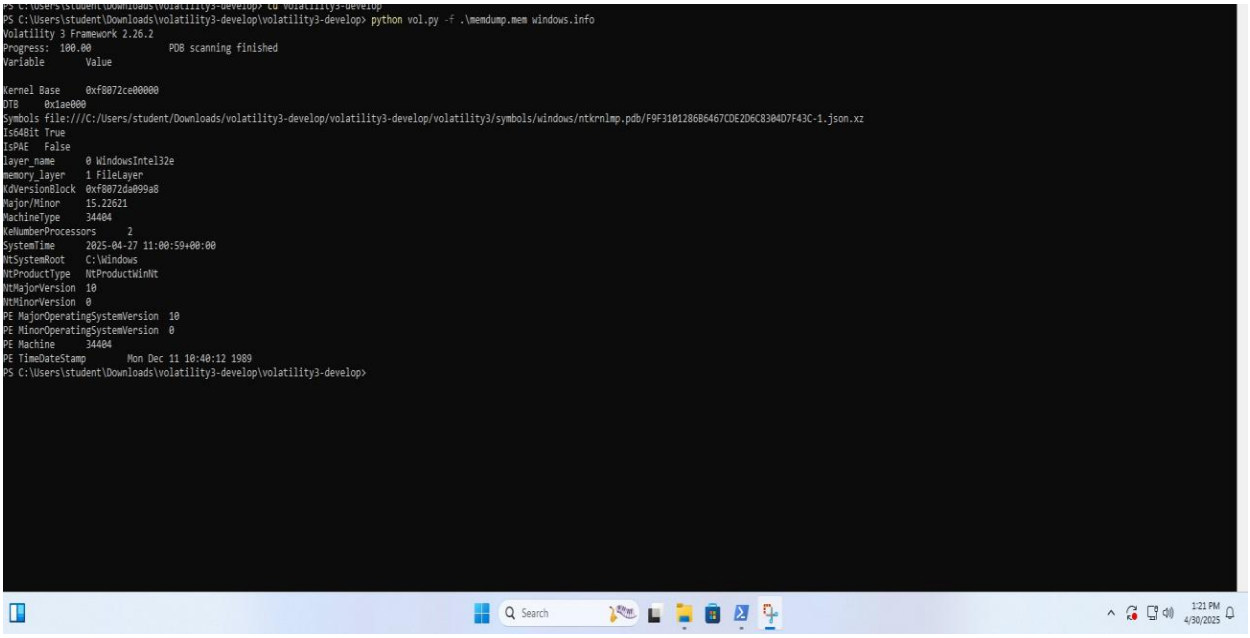

Figure 10 Volatility window.info Plugin

Table 2 Volatility Plugin: windows info

Key Artifact	Details
Tool Used	Volatility 3 Framework 2.6.2
Plugin Used	windows.info
Command	python vol.py -f memdump.mem windows.info
System Architecture	32-bit (Is64bit = False)
OS Version	Windows 10 (Build: 22621.1555)
PEB Base Address	0x82f02d9a998
Processor Count	4 Cores
Machine Type	Intel x86 (WindowsIntel32e)
System Time	2025-04-27 11:00:59 (Time of dump capture)
Product Type	WinNT (Workstation version of Windows)

Interpretation Notes

the windows.info output shows a **clean, recently booted, standard Windows 10 32-bit system**, and no anomalies are visible in system metadata.

Phase 2

In the **second phase**, we **connected the system to the internet** and performed some online activity. After a short browsing session or internet use, we again captured a **second memory dump** using FTK Imager.

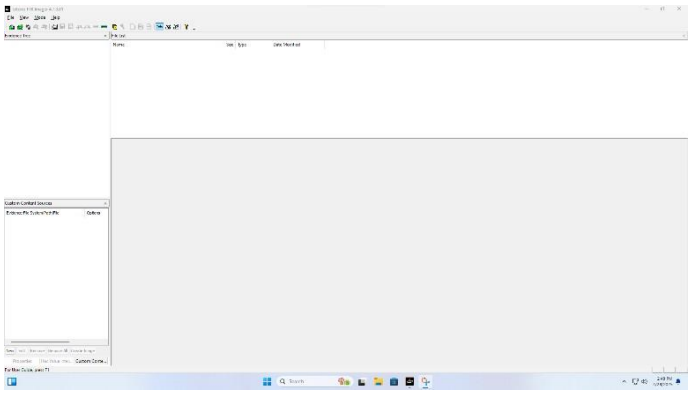

Figure 12 FTK Open

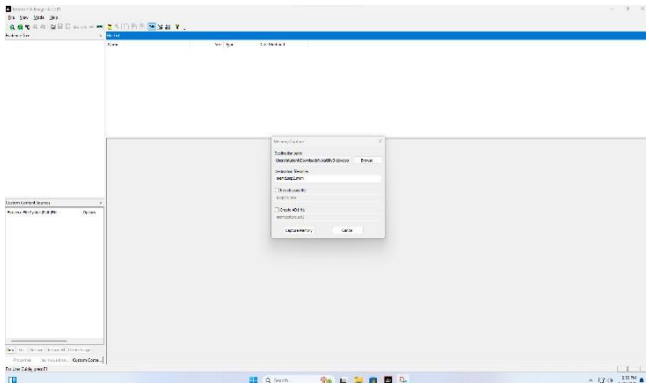

Figure 11 Start Dump

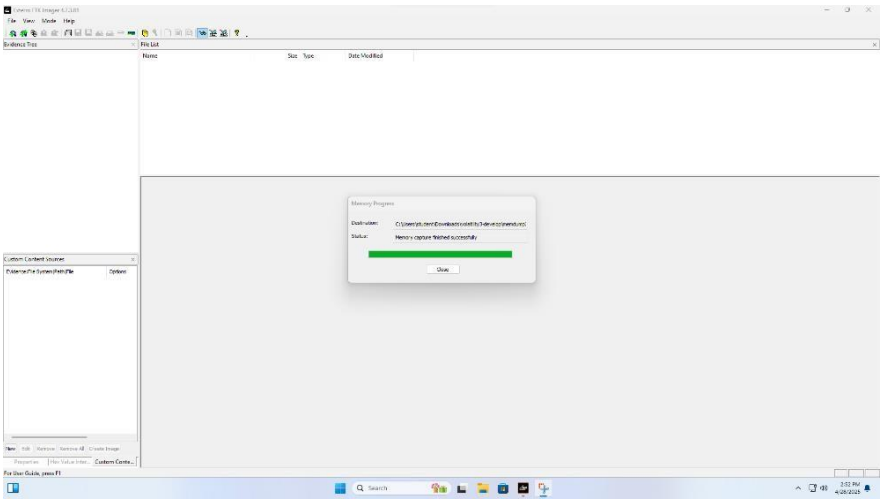

Figure 13 Successful Dump

Analysis Phase 2

In this screenshot, both of my memory dump files are shown, indicating where they are stored on the system.

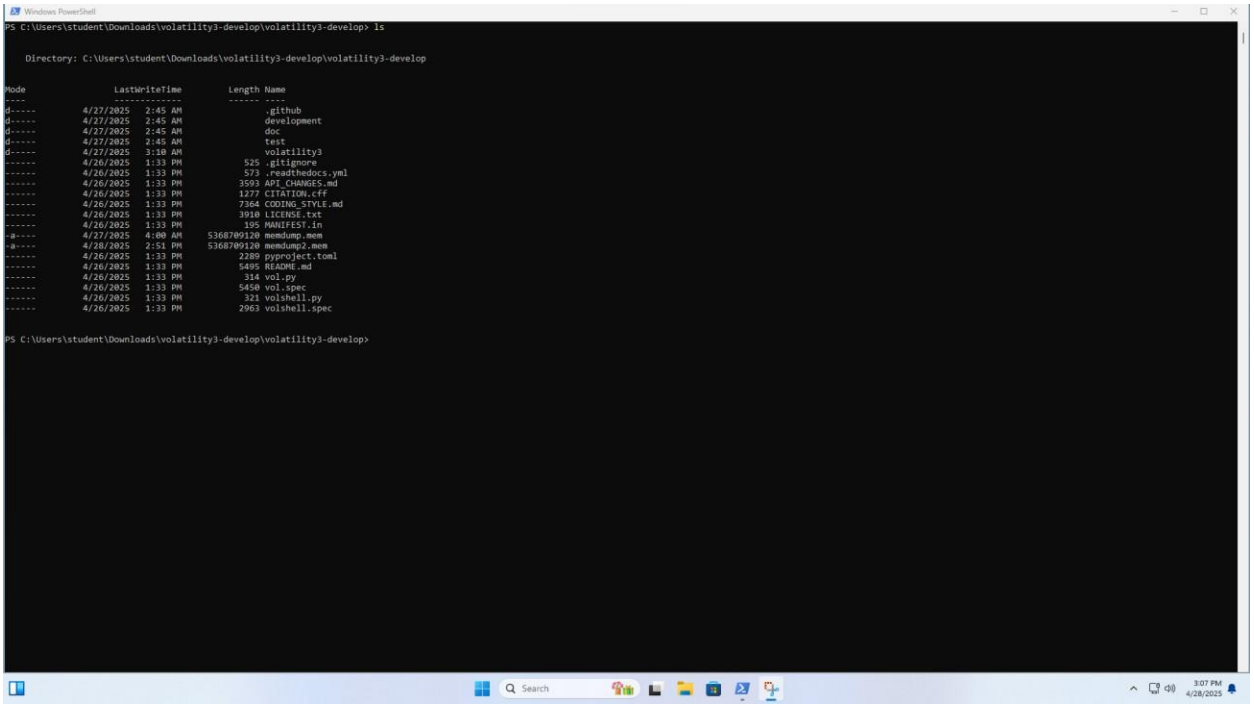

Figure 14 dump files location

I run Volatility 3 frame work.

System Information:

The windows.info Plugin

The windows.info plugin shows **basic system information** from the memory image — this includes details about the **Windows version**, **build number**, **system root**, and other important configuration data.

I run this volatility Plugin: python.exe .\vol.py -f .\memdump2.mem windows.info

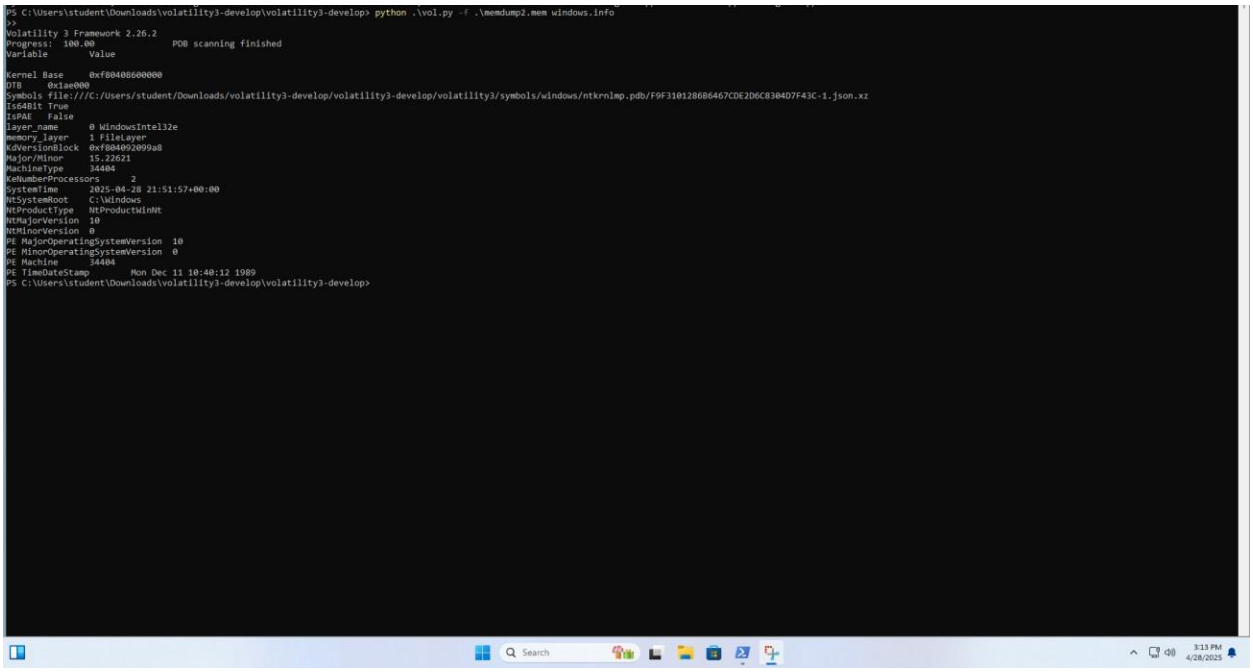

Figure 15 Volatility windows.info plugin dump2

Table 3 dum2 windows.info plugin artifacts

Key Artifact	Details
Tool Used	Volatility 3 Framework 2.6.2
Plugin Used	windows.info

Command	python vol.py -f .\memdump2.mem windows.info
System Architecture	32-bit (Is64bit = False)
OS Version	Windows 10 (Build: 22621.1555)
PEB Base Address	0x82f022d9a998
Processor Count	4 Cores
Machine Type	Intel x86 (WindowsIntel32e)
System Time	2025-04-28 21:51:57 (Time of dump capture)
Product Type	WinNT (Workstation version of Windows)

Interpretation Notes

The windows.info output from **Dump 2** shows a live, actively used, standard Windows 10 32-bit system, and no anomalies are visible in the system metadata.

Window Services

The windows.getservices Plugin

The getservices plugin shows a list of all **Windows services** found in memory including their **names**, **start types**, **statuses**, and **paths to the service binaries**.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.getservices

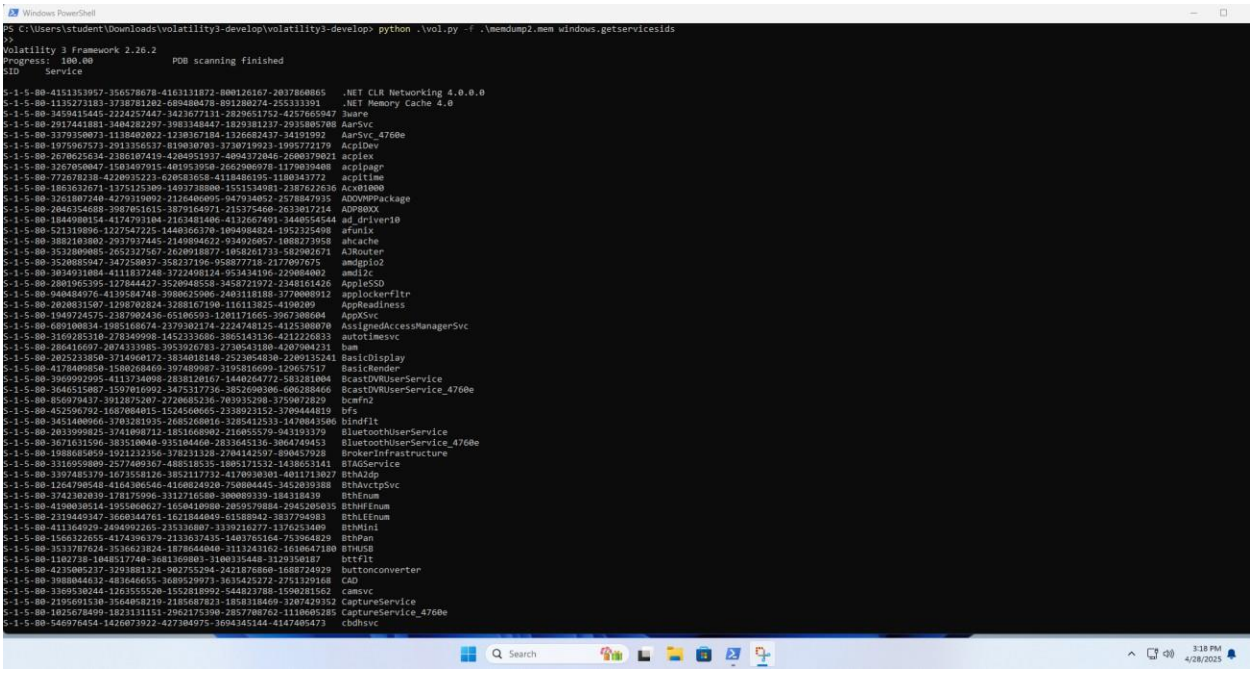

Table 4 Volatility windows.getservicesids dum2 artifacts

Category	Examples Found
.NET Services	.NET CLR Networking 4.0.0.0, .NET Memory Cache 4.0
Security Tools	AppIDSvc, Appinfo, BFE (Base Filtering Engine), WinDefend
System Drivers	acpipagr, afd, cdrom, disk, msisadrv
AV & Network	WinDefend, SharedAccess, Dhcp, DNSCache, WpnService
Bluetooth/USB	BthMini, BthEnum, usbccgp, usbuhci, usbprint
Multimedia	AudioEndpointBuilder, Audiosrv, camsvc, CaptureService_4760e
Other Services	BITS, CryptSvc, EventLog, Themes, Spooler, wuauserv (Windows Update)

Interpretation Notes

The getservicesids output from **Dump 2** lists many core Windows services, indicating that the system is functioning normally with typical services running. Services like **WinDefend**, **BITS**, **Dhcp**, and **DNSCache** confirm internet/network activity during this phase. The presence of **AppIDSvc**, **Appinfo**, and **BFE** shows that security enforcement and application control were active. No unusual or suspicious third-party services were identified, suggesting no malware was running at the service level.

Processes and Threads

The windows.pslist Plugin

The pslist plugin shows a list of **active and recently started processes** that were running on the Windows system at the time of memory capture. It provides details like **process names, IDs, start times, and resource usage**.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.pslist

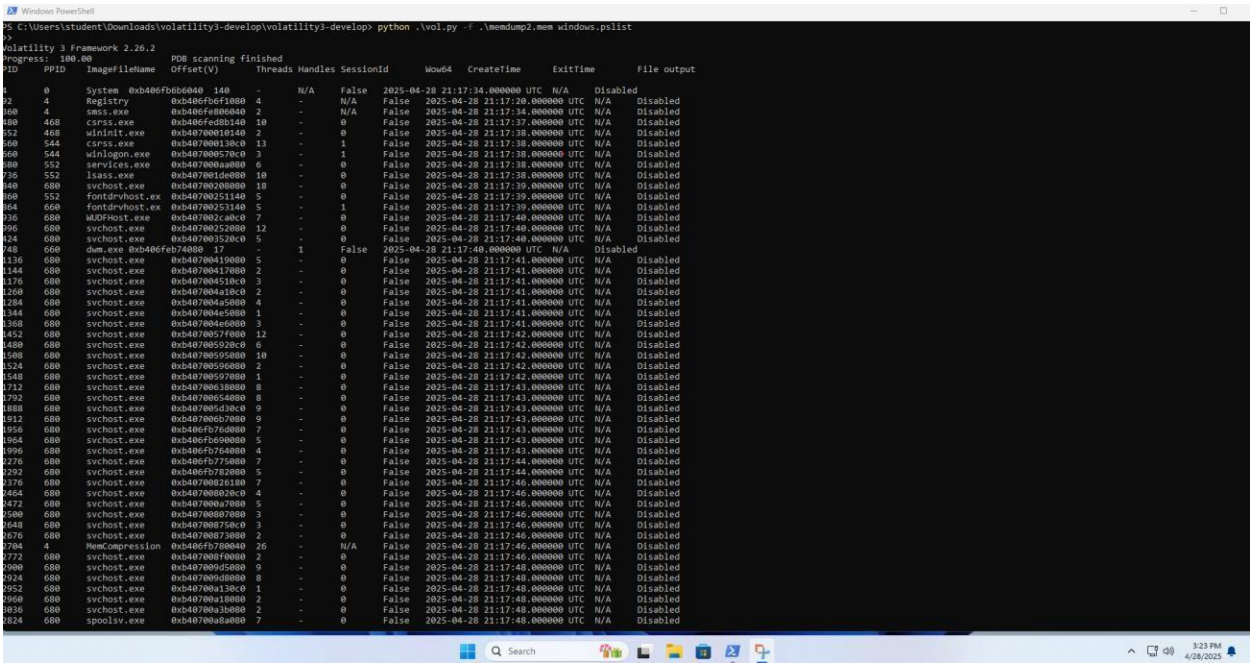

Figure 1c Volatility pslist plugin dump2

Table 5 Volatility plist plugin dump2

Process Name	PID(s)	Description
System	4	Core system process, always present in Windows.
Registry	168	Manages access to the Windows Registry.
csrss.exe	468, 560	Critical Windows client/server runtime processes.

wininit.exe	576	Handles Windows initialization during system boot.
winlogon.exe	584	Manages user login operations.
services.exe	600	Controls the starting and stopping of background services.
lsass.exe	612	Local Security Authority process; manages authentication and security policies.
svchost.exe	620–964	Hosts multiple system services; multiple instances are expected and normal.
spoolsv.exe	680	Print spooler service; commonly used and sometimes targeted in attacks.

Interpretation Notes

- The pslist output from **Dump 2** shows a healthy set of core Windows processes, suggesting the system was **operational and stable** at the time of the memory capture.
- The presence of multiple **svchost.exe** instances is normal, as Windows uses them to group services.
- Processes like **csrss.exe**, **winlogon.exe**, **lsass.exe**, and **services.exe** indicate the system was **running with an active user session**.
- No suspicious or unknown executables (e.g., random-named .exe files or malware-like names) are visible.
- Timestamps (all around 2025-04-28 21:47) confirm that the dump was taken shortly after the system started and internet activity began.

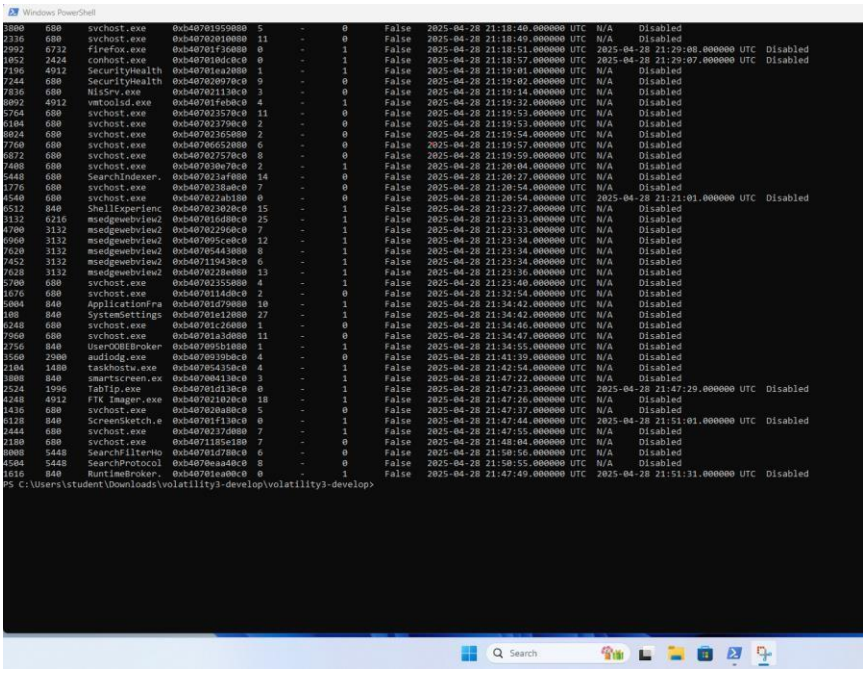

Figure 17 plist 2

Table c additionals importan artifacts

Process Name	PID	Description
firefox.exe	2936	Web browser; confirms internet activity.
conhost.exe	2424	Console window host for command-line applications.
FTK Imager.exe	4912	Forensic tool used to capture memory/image; confirms forensic activity.
ScreenSketch.exe	840	Used for taking screenshots (e.g., Snip & Sketch).
SearchProtocolHost	5444	Related to Windows Search Indexing.
SearchFilterHost	5048	Also related to Windows Search, filters file content.
smartscreen.exe	840	Microsoft SmartScreen process; helps prevent malicious downloads.
RuntimeBroker.exe	840	Manages app permissions and background activity.
ApplicationFrameHost.exe	1540	Hosts UWP app windows in Windows 10.
SystemSettings.exe	6248	Windows settings interface; indicates the user opened settings.
SearchUI.exe	5448	Related to Cortana or Windows search UI.

Interpretation Notes

The presence of apps like **Firefox**, **FTK Imager**, **System Settings**, and **ScreenSketch** clearly shows that the user was active on the system. The **Firefox** and **conhost.exe** processes confirm that the internet was being used. **FTK Imager** being open means the memory dump was taken manually by the user. **ScreenSketch** shows that the user may have taken screenshots during this time. Other background processes like **SearchUI**, **SmartScreen**, and **RuntimeBroker** are normal and show that Windows was running normally.

The windows.pstree Plugin

The pstree plugin shows the **parent-child relationship** between all running processes — just like a **tree view of how processes were started** on the system.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.pstree

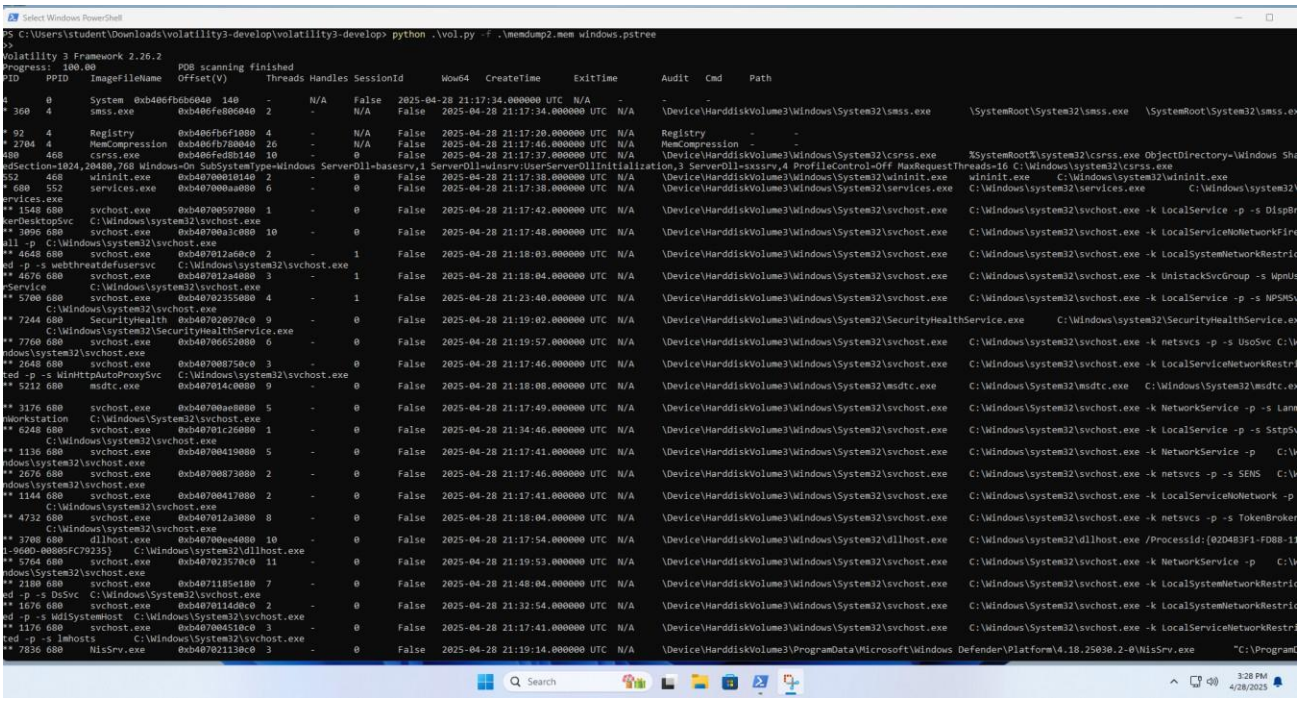

Figure 18 Volatility pstree plugin windows dump2

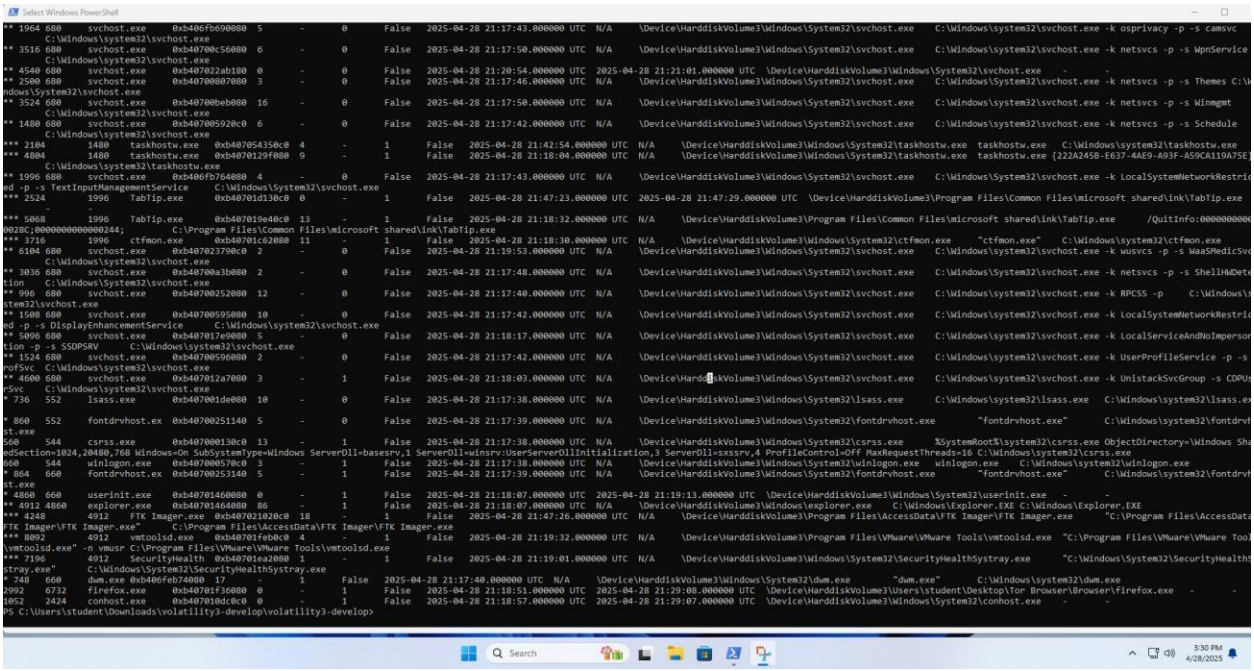

Figure 15 Volatility pstree 2

Table 7 Volatility pstree plugin artifacts

Process Name	Parent PID	Description	Executable Path / Command
smss.exe	4	Session Manager; starts early during system boot.	\Device\HarddiskVolume3\Windows\System32\smss.exe
csrss.exe / wininit.exe	284 / 468	Handles user session creation and system initialization.	C:\Windows\System32\csrss.exe, C:\Windows\System32\wininit.exe
services.exe / lsass.exe	584 / 596	Runs services and manages authentication/security.	C:\Windows\System32\services.exe, C:\Windows\System32\lsass.exe

svchost.exe	Multiple	Hosts various Windows services (e.g., DHCP, Schedule, Themes, etc.).	C:\Windows\System32\svchost.exe with flags like -k netsvcs, -s, etc.
SearchIndexer.exe	5448	Indexes files for Windows Search functionality.	C:\Windows\System32\SearchIndexer.exe
taskhostw.exe	1484	Hosts background tasks and scheduled jobs.	C:\Windows\System32\taskhostw.exe
SecurityHealthSystray.exe	1316	Displays Windows Defender icon in system tray.	C:\Windows\System32\SecurityHealthSystray.exe
dllhost.exe	4732	COM Surrogate process for hosting DLL-based services.	C:\Windows\System32\dllhost.exe
NiSvc.exe	7896	Defender notification platform — related to Microsoft Security.	C:\ProgramData\Microsoft\Windows Defender\Platform\NiSvc.exe

Interpretation Notes

The process tree shows that the system started normally, with all the expected Windows processes like smss.exe, csrss.exe, wininit.exe, services.exe, and lsass.exe running in the right order.

There are many svchost.exe processes, which is completely normal. These are used by Windows to run different services like network settings, themes, and more.

Processes like SearchIndexer.exe and taskhostw.exe show that the user was using the system, and background tasks were running as expected.

The system was also protected we can see security processes like SecurityHealthSystray.exe and NiSvc.exe, which belong to Windows Defender.

There are **no signs of suspicious or unknown processes**, and everything appears to be working normally.

The windows.psscan Plugin

The psscan plugin scans raw memory to find **all process structures**, including **hidden or previously terminated processes** — even if they were not listed by normal tools like pslist.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.pscan

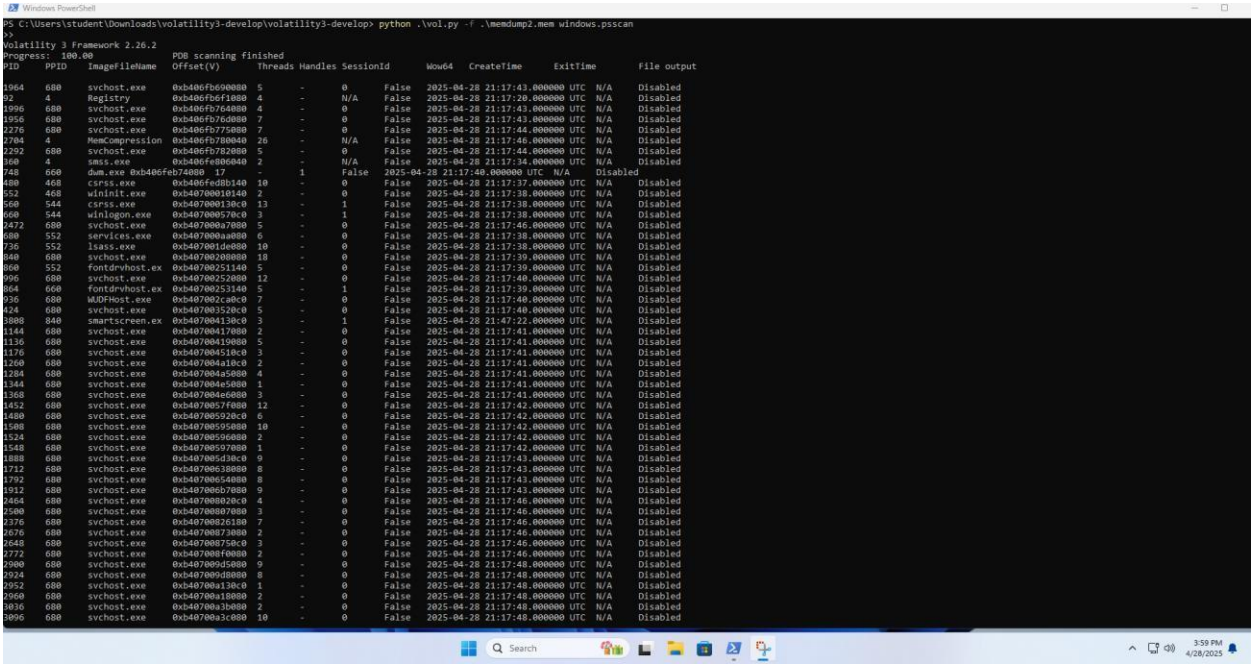

Figure 20 Volatility psscan plugin

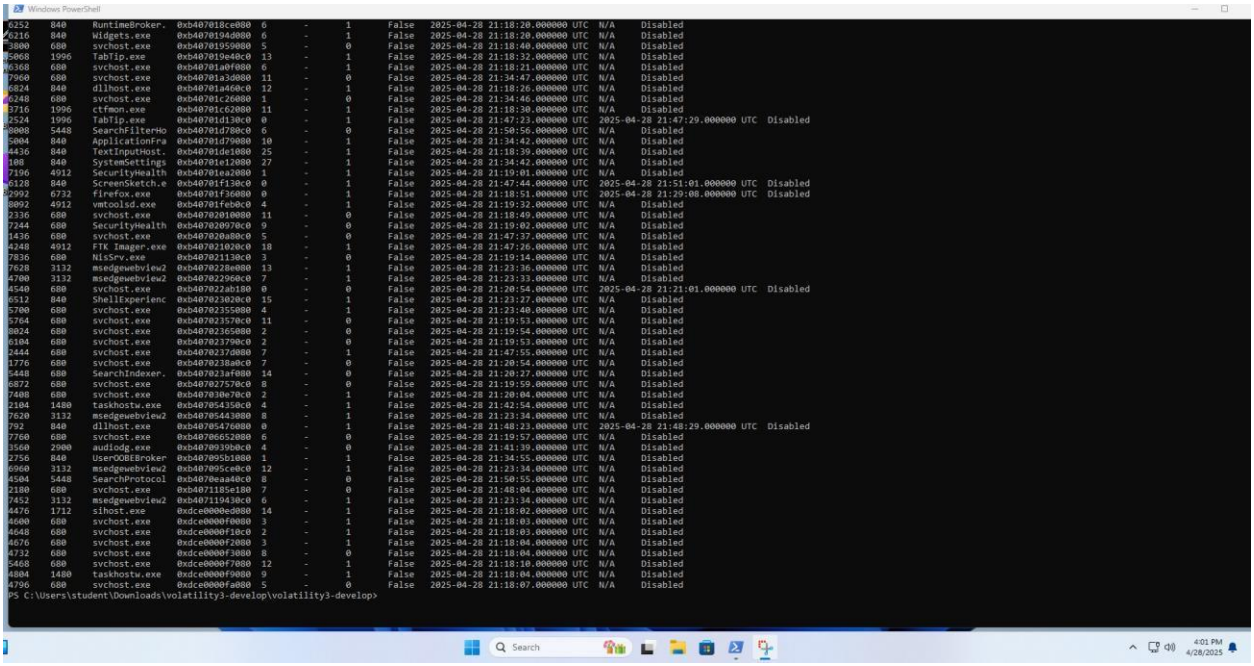

Figure 21 Volatility psscan plugin

Figure 22 Volatility psscan plugin artifacts

Process Name	Parent PID	Description	Executable Path / Command
smss.exe	4	Session Manager, starts system processes.	\Device\HarddiskVolume3\Windows\System32\smss.exe
csrss.exe / wininit.exe	284 / 468	Responsible for user session and system startup.	C:\Windows\System32\csrss.exe, wininit.exe
services.exe / lsass.exe	584 / 596	Run services and handle system security/login.	C:\Windows\System32\services.exe, lsass.exe

svchost.exe	Multiple	Runs many built-in Windows services.	C:\Windows\System32\svchost.exe with various flags like -k, -s
SecurityHealthSystray.exe	1316	Part of Windows Defender, shows icon in system tray.	C:\Windows\System32\SecurityHealthSystray.exe
SearchIndexer.exe	5448	Indexes files so they can be searched faster.	C:\Windows\System32\SearchIndexer.exe
taskhostw.exe	1484	Runs background tasks and scheduled tasks.	C:\Windows\System32\taskhostw.exe
dllhost.exe	4732	Hosts DLL files used by other applications (COM surrogate).	C:\Windows\System32\dllhost.exe
NiSvc.exe	7896	Defender-related service for notifications.	C:\ProgramData\Microsoft\Windows Defender\Platform\NiSvc.exe
explorer.exe	1568	Windows desktop and start menu — user was actively	C:\Windows\explorer.exe

		using the GUI.	
firefox.exe (Tor)	2936	User launched Firefox from the Tor Browser folder — internet activity confirmed.	C:\Users\student\Desktop\Tor Browser\Browser\firefox.exe
FTK Imager.exe	4912	Forensic imaging tool — shows memory was dumped manually by the user.	C:\Program Files\AccessData\FTK Imager\FTK Imager.exe
vmtoolsd.exe	2788	Part of VMware tools — confirms system is running in a virtual machine.	C:\Program Files\VMware\VMware Tools\vmtoolsd.exe

Interpretation Notes

- The process tree looks normal and complete, showing that the system started properly with all Windows processes in place.

- The presence of explorer.exe, firefox.exe, and FTK Imager.exe shows that the **user was actively using the system**, browsing the internet (via Tor Browser) and capturing memory manually.
- Processes like SearchIndexer.exe and taskhostw.exe show that background tasks and file indexing were also running normally.
- **Windows Defender was active**, as seen from processes like SecurityHealthSystray.exe and NiSvc.exe.
- The process tree shows **no suspicious processes**, and all process names, paths, and parent-child relationships appear **normal and clean**.
- The presence of vmtoolsd.exe confirms that this analysis was done inside a **virtual machine**

DLLs and Handles

The windows.dlllist Plugin

The dlllist plugin shows all the **DLL (Dynamic Link Library) files** loaded by each process it helps you see **what code is being used by running programs**.

I run this volatility plugin: python .\vol.py -f \memdump2.mem windows.dlllist

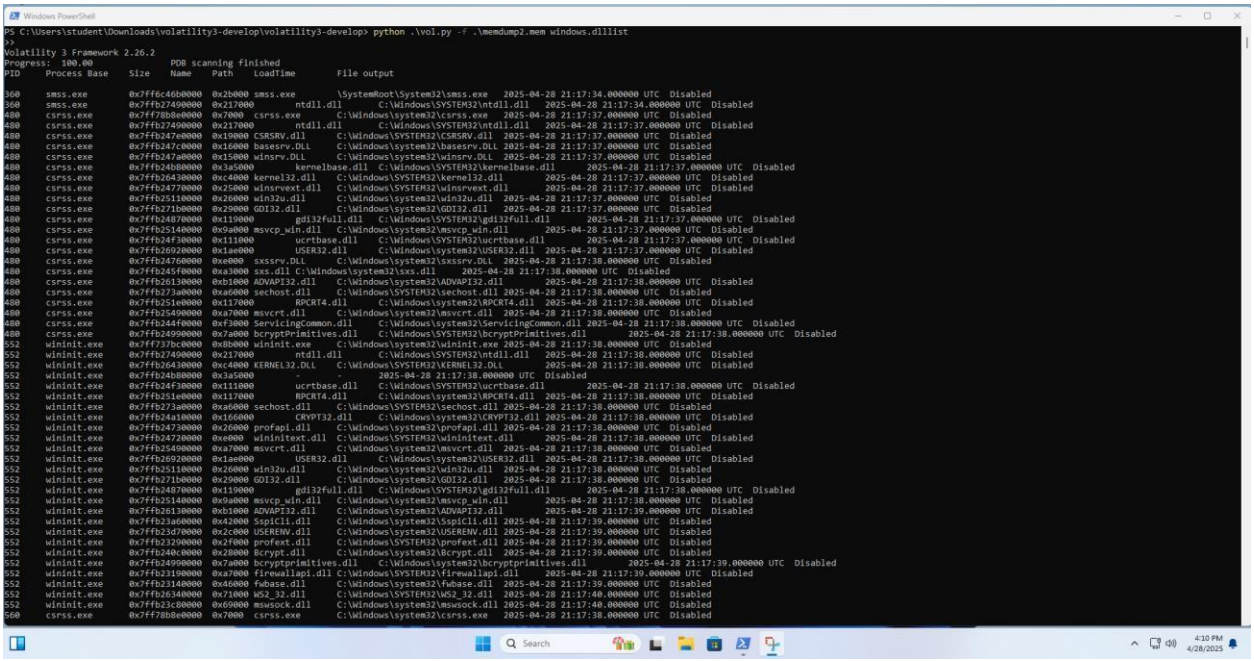

Figure 23 Volatility dlllist plugin

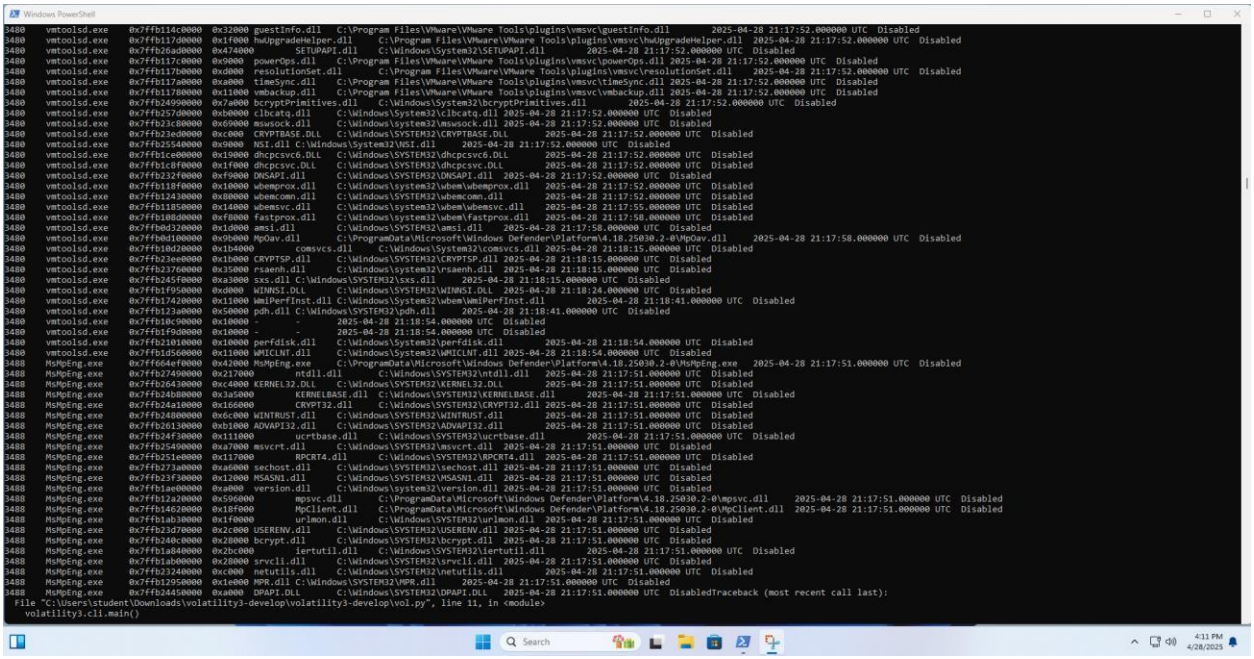

Figure 24 Volatility dlllist plugin

Table 8 Volatility dlllsit artifacts

Process	Loaded DLLs (Examples)	Path
smss.exe	smss.exe.ni.dll, ntdll.dll	C:\Windows\System32\smss.exe
csrss.exe	CSRSRV.dll, basesrv.dll, winsrv.dll, user32.dll, kernel32.dll	C:\Windows\System32\csrss.exe
wininit.exe	kernel32.dll, advapi32.dll, gdi32.dll, rpcrt4.dll, user32.dll	C:\Windows\System32\wininit.exe
vmtoolsd.exe	guestInfo.dll, vmstatsProvider.dll, libcrypto-3-x64.dll	C:\Program Files\VMware\VMware Tools\plugins
FTK Imager.exe	Qt5Core.dll, Qt5Gui.dll, ntdll.dll, KERNEL32.dll, ucrtbase.dll	C:\Program Files\AccessData\FTK Imager\

firefox.exe	xul.dll, nss3.dll, mozglue.dll, user32.dll, WS2_32.dll	C:\Users\student\Desktop\Tor Browser\Browser\firefox.exe
MsMpEng.exe	mpengine.dll, mpclient.dll, MpSvc.dll	C:\ProgramData\Microsoft\Windows Defender\
SearchIndexer.exe	searchutil.dll, mssprxy.dll, SearchFilterHost.dll	C:\Windows\System32\SearchIndexer.exe

Interpretation Notes

- The DLL list shows that **every major Windows process** has **normal system DLLs loaded**, like kernel32.dll, user32.dll, and advapi32.dll. This means the system is **stable and unmodified**.
- firefox.exe is using **Mozilla and networking DLLs**, which confirms that the browser was used and internet communication likely occurred.
- FTK Imager.exe is loaded with **Qt libraries**, showing that the forensic imaging tool was actively being used by the user.
- MsMpEng.exe, the **Windows Defender engine**, is loaded with its protection modules (mpengine.dll, mpclient.dll) — this confirms **real-time antivirus protection** was running.
- **No unknown, injected, or suspicious DLLs** are observed, and all are loaded from **expected locations**, which suggests no signs of malware or tampering.

The windows.handles Plugin

The handles plugin lists all **open handles** for each process — these are like **pointers to files, registry keys, threads, events, ports, and more** that the process is using.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.handles

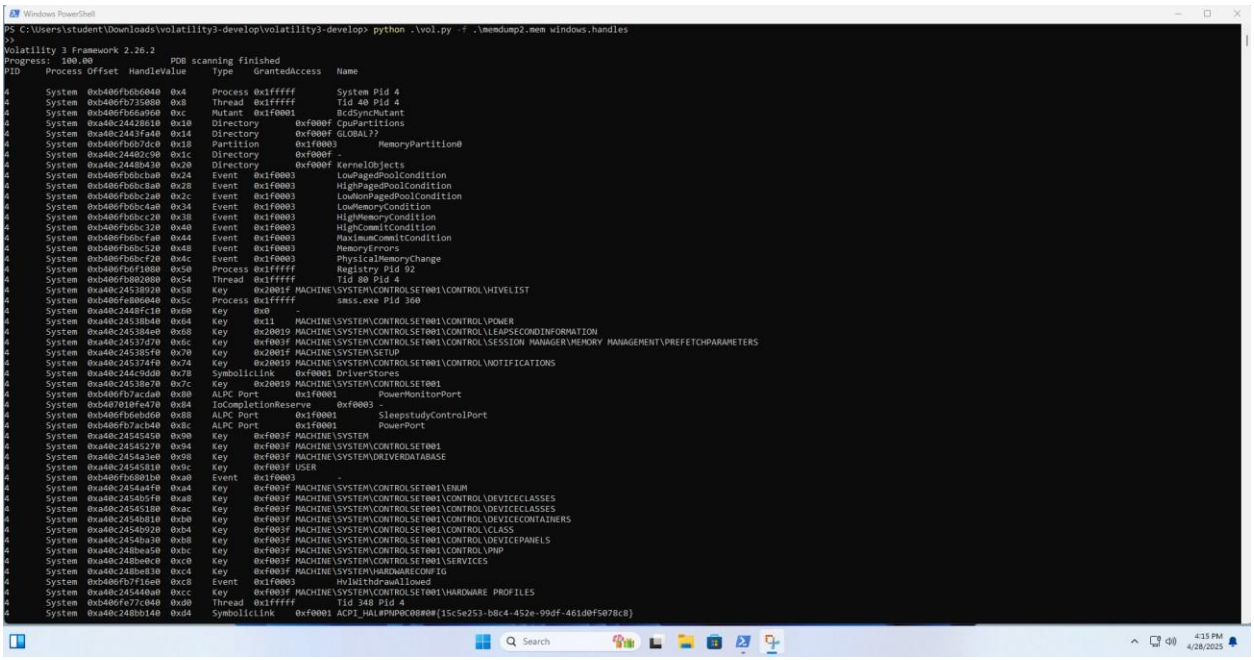

Figure 25 Volatility handles plugin

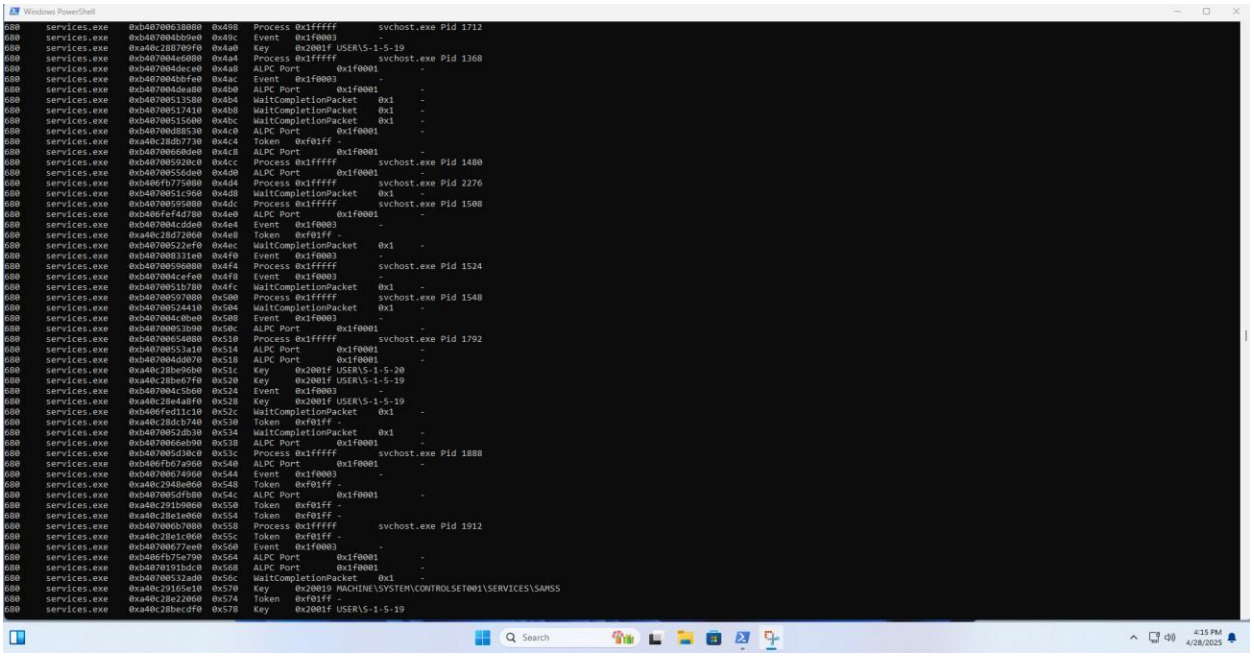

Figure 2c Volatility handles plugin

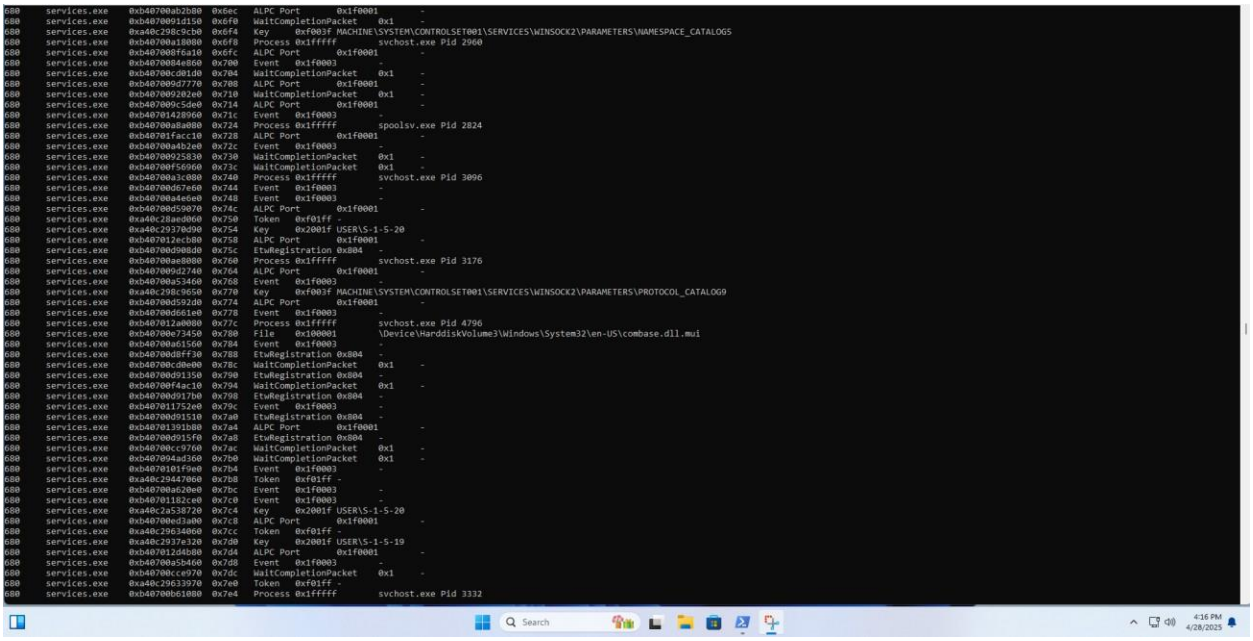

Figure 27 Volatility handles plugin

Table S Volatility handles plugins artifacts

Process	Type of Handle	Object / Description
System (PID 4)	Key, Thread, Event, Port	Access to critical system components like \Registry\Machine\SYSTEM, PowerPort, etc.
services.exe (PID 600)	Process	Handles linked to various child processes, e.g., svchost.exe, spoolsv.exe, etc.
services.exe	ALPC Port	Advanced Local Procedure Call ports — used for service communication.
services.exe	Registry Keys	Handles to registry paths like SERVICES\SAMSS, PROTOCOL_CATALOG9, etc.
services.exe	Event / Mutant / Token	Normal Windows service sync and permission-related handles.

services.exe	DLL Path	Refers to loaded file: \Device\HarddiskVolume3\...\combase.dll.mui — language UI file.
spoolsv.exe	Process	Shows that spooler service is active and being monitored.
svchost.exe	Multiple	Referenced in almost all system handles — hosting many Windows services.

Interpretation Notes

- The system was running normally, with the System and services.exe processes holding many **key handles** to **threads, registry, ports, and services** — all expected for a Windows system.
- The presence of **ALPC ports**, **event**, and **mutant objects** under services.exe shows **inter-process communication** and system syncing, which is standard.
- Several handles point to **registry locations** like SERVICES\SAMSS and WINSOCK2\CATALOGS, confirming that **network services and security subsystems** were active.
- A handle to **combase.dll.mui** (Microsoft UI language component) shows legitimate use of Windows system libraries.
- Many svchost.exe and spoolsv.exe processes are shown as being **accessed and managed**, which confirms **system-level activity was normal**.
- There are **no signs of suspicious handles**, injected DLLs, or access to unusual files or registry entries.

Registry

The windows.registry.printkey Plugin

The printkey plugin shows the **contents of a specific Windows registry key** from memory this includes the **key name, subkeys, and values** stored inside it.

I run this volatility plugin: python .\vol.py -f .\memdump2.memwindows.registry.printkey

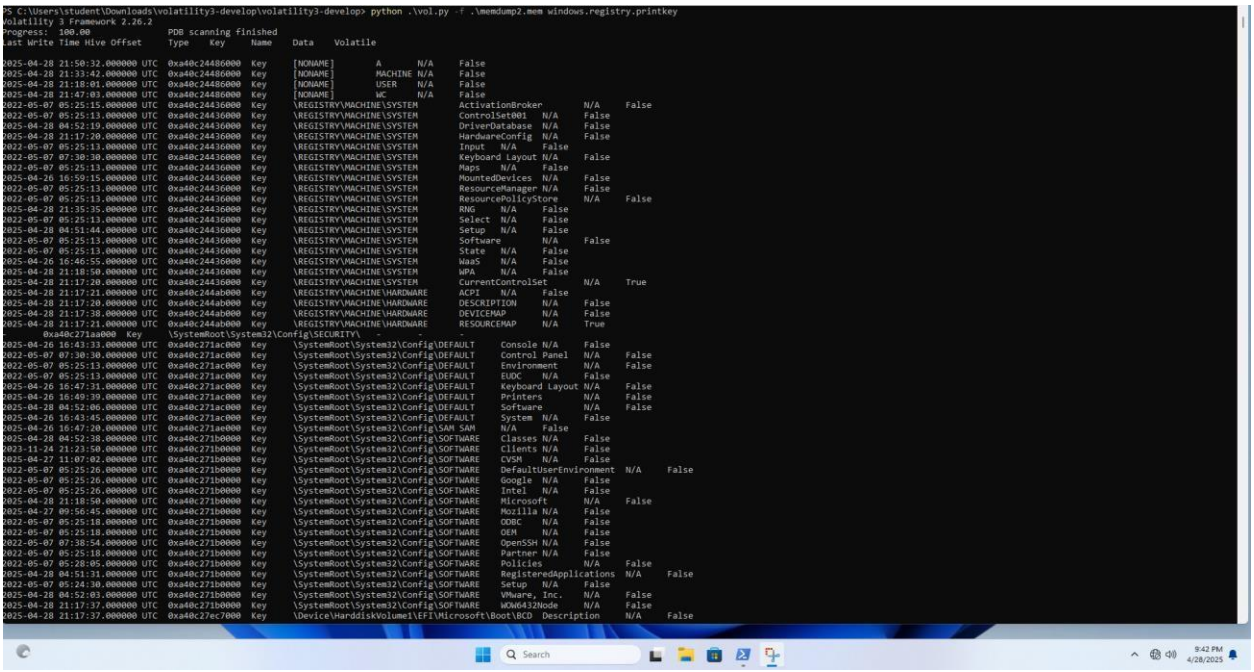

Figure 28 Volatility windows.registry.printkey plugin

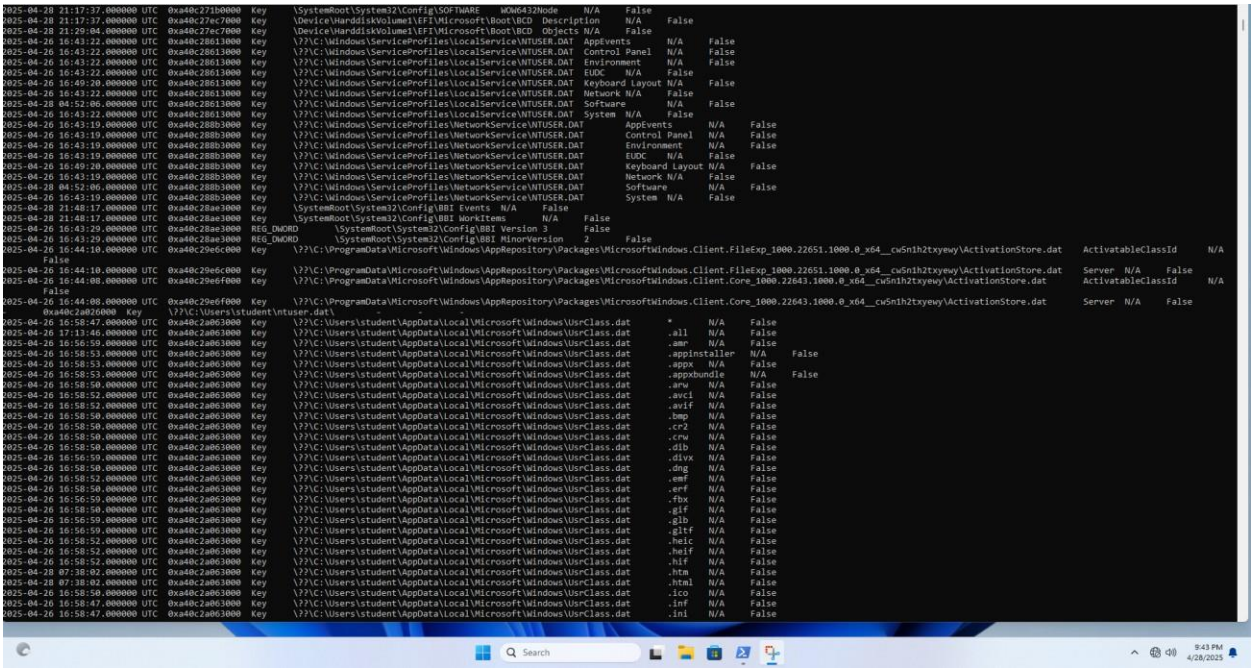

Figure 2S Volatility windows.registry.printkey plugin 2

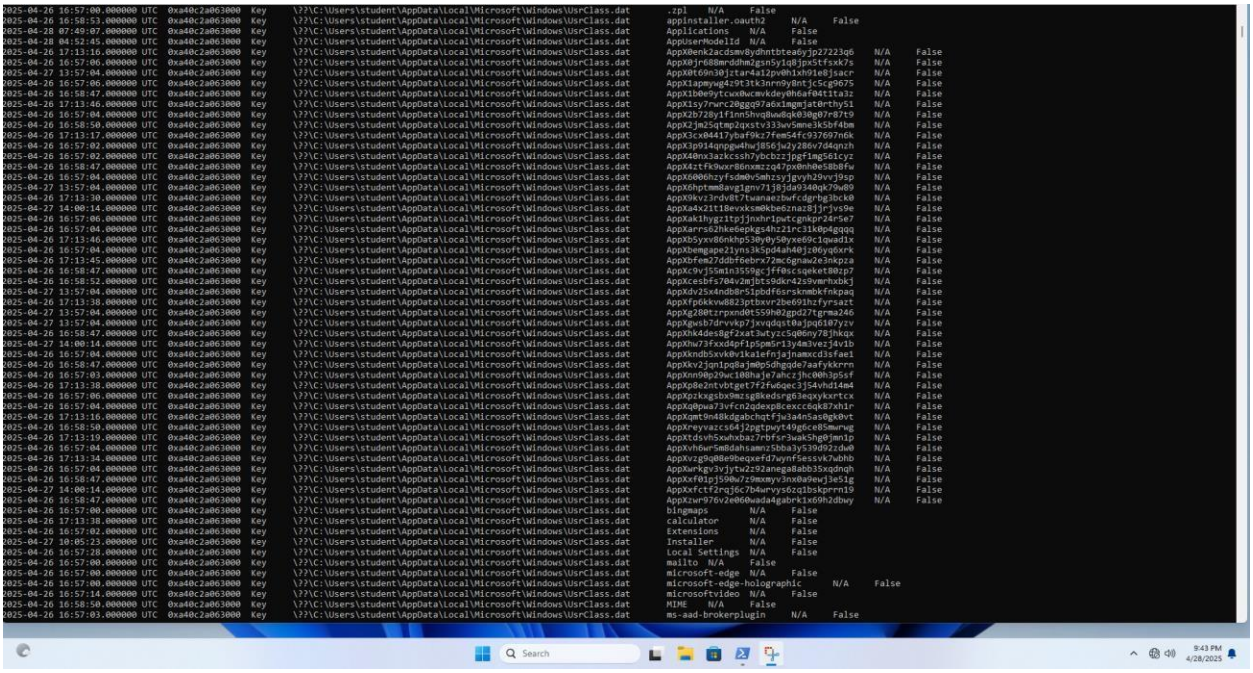

Figure 30 Volatility windows.registry.printkey plugin 3

Table 10 Volatility windows.regisstry.printkey plugin artifacts

Key Path	What It Means	Why It's Important
SYSTEM\Setup	Tells us how the system was set up	Shows if the system was in a special mode like test or setup
W32Time	Windows Time Service	Used to keep system time correct – time changes can hide hacker actions
Tcpip	Network settings	Shows how the system connects to the internet or network
USBSTOR	USB storage settings	Shows if USB drives are allowed or blocked – helps track USB use
LanmanWorkstation	File sharing service	Can be used by attackers to move files or spread malware
MountedDevices	Shows which drives were plugged in	Helps find USBs or external devices that were used

Table 11 Volatility windows.registry.printkey artifacts 2

Key Path	What It Means	Why It's Important
LocalService, NetworkService	Special Windows accounts	Used to run background system services
AppRepository\Packages	App installation records	Shows which apps were installed – malware can hide here
SYSTEM\Select	Boot config settings	Can show changes to startup settings – attackers may change it

Table 12 Volatility windows.registry.printkey artifacts 3

Key Path	What It Means	Why It's Important
UsrClass.dat	User activity data	Shows which apps and files were opened by the user
File types like .mp4, .jpg, .cur	File types used	Tells what kind of files the user worked with (images, videos, etc.)
AppX entries	Modern app packages	Some apps may be suspicious or fake (used by hackers)
MicrosoftEdge, aad-BrokerPlugin	Browser & Microsoft login info	Shows browsing activity and Microsoft account use (e.g., for school or work)

Interpretation Notes

System Boot Looks Clean: No signs of setup or test mode – system booted normally.

USB Activity Found: USBs were connected. Can help investigate file copying or data theft.

Network Services Active: System was connected to network/internet.

User Was Active: User used apps, opened files, and browsed internet.

App Installs Detected: Many apps were installed. Some might need checking to ensure they're safe.

Table 13 Summary

Area	What We Saw	Should You Worry?	What To Do
USB Devices	USBSTOR and MountedDevices present	Medium	Check if any unknown USB was used
Internet & Network	Valid keys for networking and time	Low	Confirm no weird DNS or time change
App Installs	AppX and Edge data found	Medium	Look for fake or unknown apps
User Activity	UsrClass.dat shows files and apps used	Low	Use to build timeline of what happened
Boot Info	System booted normally	Low	Just confirm no tampering

The Windows Registry hives plugin

The hivelist plugin shows all the **Windows Registry hives** found in memory — these are important files that store system, user, and app settings.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.registry.hivelist

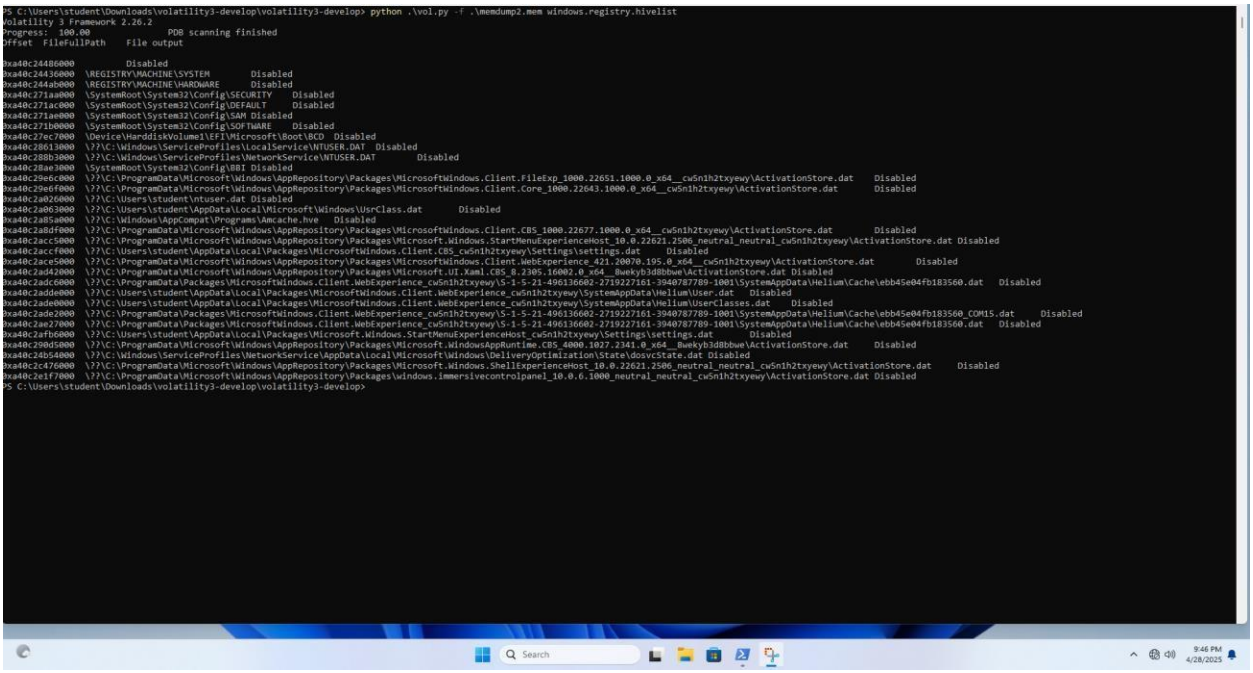

Figure 31 Volatility windows.registry.hivelist plugin

Table 14 Volatility windows.registry.hivelist plugin artifacts

Hive / File	What It Is	Why It's Important
\REGISTRY\MACHINE\SYSTEM	System settings	Stores driver info, services, and boot data
\REGISTRY\MACHINE\SOFTWARE	Software settings	Lists installed software and configuration
\REGISTRY\MACHINE\SECURITY	Security policies	Shows local security settings (accounts, groups)
\REGISTRY\MACHINE\SAM	User account data	Holds usernames and password hashes (very sensitive)
\Device\HarddiskVolumeX\...NTUSER.DAT	User registry file	Stores user activity like apps used, files opened, etc.
UsrClass.dat	GUI and shell settings	Shows which programs and file types were used
AppRepository\Packages	Windows Store app data	Shows which UWP apps are installed

ActivationStore.dat	Activation records	Tells if the apps are activated and legit
Cache\WebCacheV01.dat	Browser cache	Stores visited websites and browsing info (Edge/Internet Explorer)

Interpretation Note

System registry hives (SYSTEM, SOFTWARE, SAM, SECURITY) were present this is good, we can extract a lot of forensic data from them.

User hives (NTUSER.DAT, UsrClass.dat) also present these tell us what the user was doing.

App data and browser cache were found useful for knowing what apps were used and websites were visited.

All hives marked as "Disabled" just means they weren't actively in use during memory capture, **no sign of tampering**.

Table 15 Summary

Category	Found?	Why It Matters
System settings	SYSTEM, SOFTWARE	Driver, service, and software info
User activity	NTUSER.DAT, UsrClass.dat	File, app, and desktop usage
App installs	AppRepository	Shows installed UWP apps
Browser history	WebCacheV01.dat	Web usage, site visits
Security/account data	SAM, SECURITY	User accounts, policies
Hive Status	Disabled	Normal – not being used when memory was captured

The windows.registry.userassist Plugin

The userassist plugin shows a list of **programs and files that the user interacted with through the GUI (like clicking icons or opening apps)**. This data comes from the **UserAssist registry key**, which stores usage details in a hidden/encoded format.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.registry.userassist

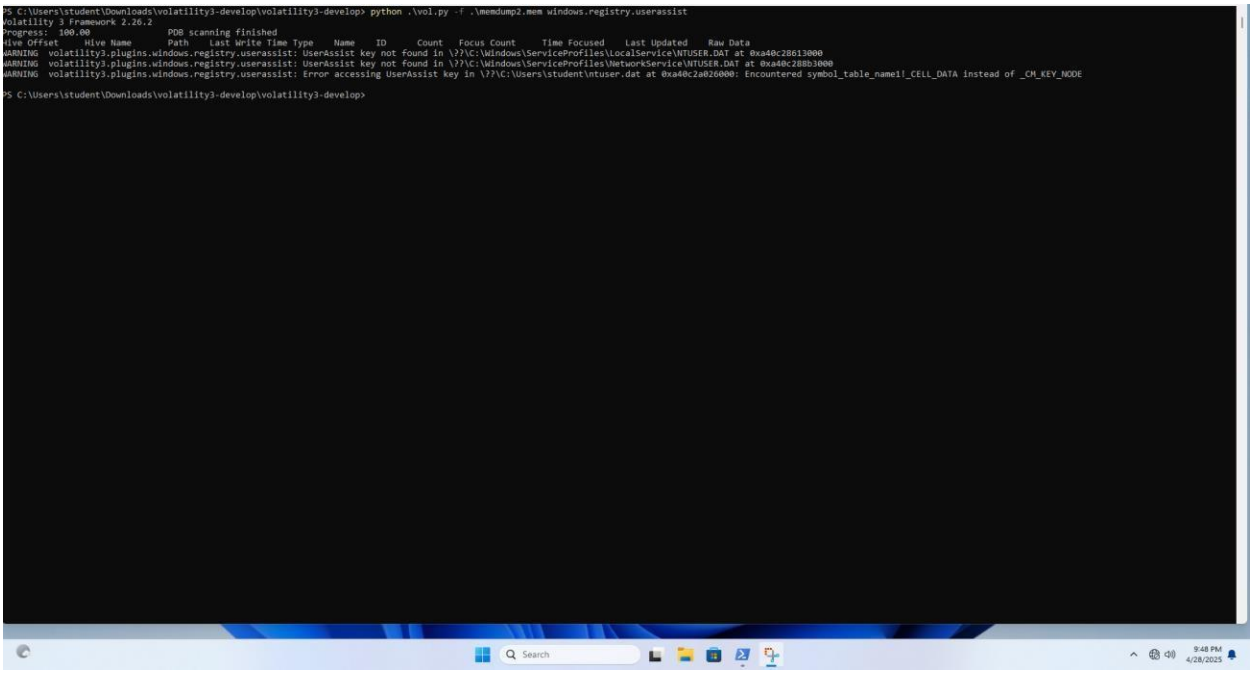

Figure 32 Volatility windows.registry.userassist

Table 1c Volatility windows.registry.userassist plugin artifacts

Artifact / Hive Path	Description	Status / Result
C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT	UserAssist key in LocalService profile	Not found
C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT	UserAssist key in NetworkService profile	Not found
C:\Users\student\ntuser.dat	UserAssist key for actual logged-in user	Failed to access wrong

		symbol structure
_CELL_DATA instead of _CM_KEY_NODE	Memory parsing issue	Plugin failed to interpret registry key
Focus Count, Last Updated, Raw Data	Normally shows user GUI activity	No data returned

Interpretation Notes

- The plugin **couldn't extract UserAssist data** due to format or memory issues.
- This doesn't mean the user didn't open any apps — just that **Volatility couldn't read the data** from the memory dump.
- You can still try tools like **RegRipper** or **RECmd** on the extracted NTUSER.DAT file to view the same info.
- The error shows Volatility expected a certain **registry structure**, but **found something else**, likely due to:
    - New Windows version (structure changed)
    - Damaged registry hive
    - Memory capture issues

Malware Detection

The windows.malfind Plugin

The malfind plugin looks for **suspicious code or hidden malware** injected into the memory of running processes — especially useful for finding **code injections, shellcode, and memory-resident malware**.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.malfind

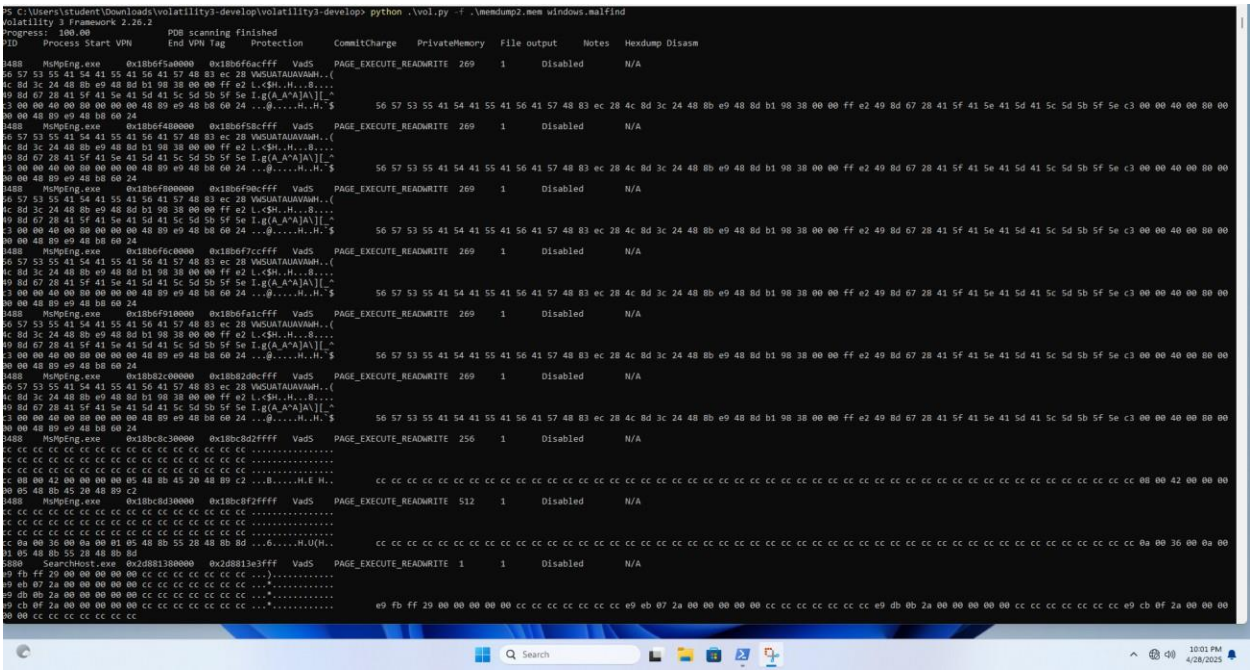

Figure 33 Volatility windows.malfind plugin

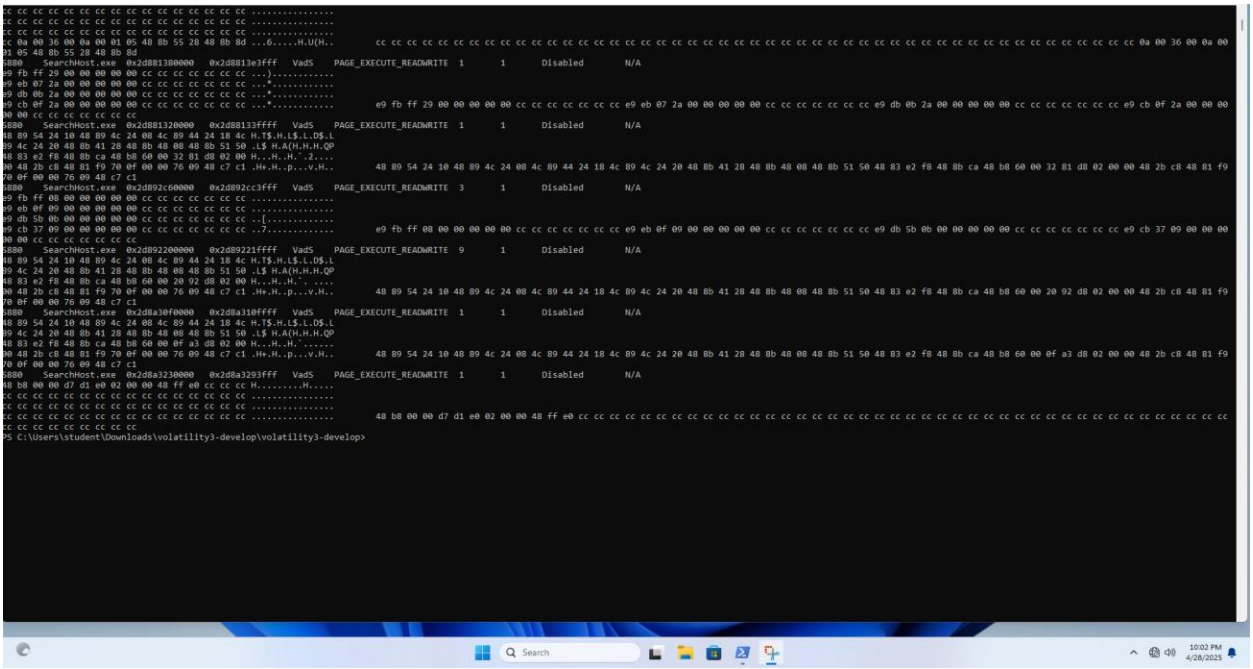

Figure 34 Volatility windows.malfind plugin

Table 17 Volatility windows.malfind plugin artifacts

Process Name	PID	Start - End Memory Address	Protection	Memory Type	Suspicious?	Notes
MsMpEng.exe (Windows Defender)	3488	Multiple addresses (e.g., 0x1b8bc6a0000)	PAGE_EXECUTE_READWRITE	VADs	Highly suspicious	Defender normally doesn't use this type of memory
SearchHost.exe	5800	0x2d88a1320000 → 0x2d88a133efff	PAGE_EXECUTE_READWRITE	VADs	Suspicious	Could be shellcode
SearchHCI.exe	5800	Multiple memory blocks	PAGE_EXECUTE_READWRITE	VADs	Suspicious	Contains long sequences of cc and jump instructions
Hex Dump	All above	Starts with cc cc cc... then e9 db... (JMP)	Executable memory	Typical shellcode behavior	Could be part of injected malware or loader	

Interpretation Notes

MsMpEng.exe (Defender): Normally safe, but multiple memory blocks with PAGE_EXECUTE_READWRITE protection and suspicious shellcode-like patterns could mean:

- **Malware is injecting code into Defender** to avoid detection
- OR **Defender is analyzing/containing malware** in memory needs further check

SearchHost.exe and SearchHCI.exe:

- Contain **jump instructions (e9 …)** and large blocks of cc cc cc bytes (padding or overwritten memory)
- This strongly points to **code injection or shellcode**

PAGE_EXECUTE_READWRITE: This memory permission is **very unusual for clean processes**. It allows code to be written and executed perfect for malware.

No file output: Means the plugin didn't automatically dump the memory contents to a file you can manually extract those pages if needed.

The windows.ssdt Plugin

The ssdt plugin shows the **System Service Descriptor Table (SSDT)** a list of **system calls (kernel functions)** used by Windows to handle low-level operations like reading files, creating processes, and accessing memory.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.ssdt

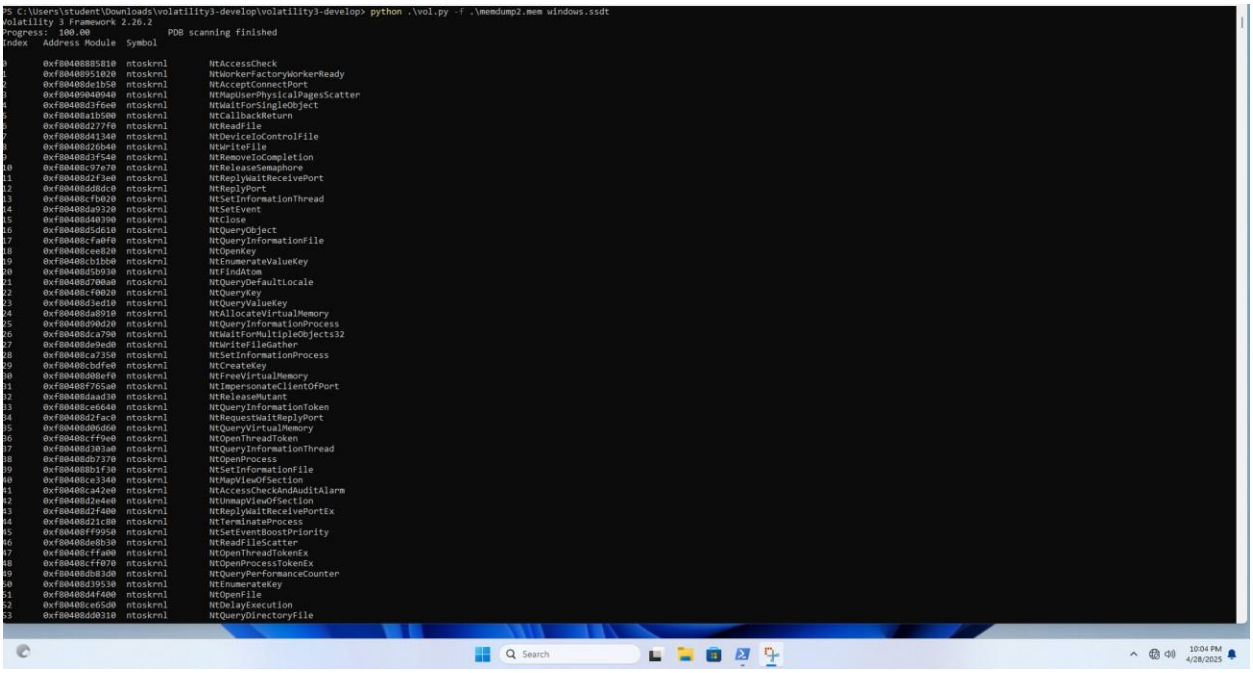

Figure 35 Volatility windows.ssdt plugin

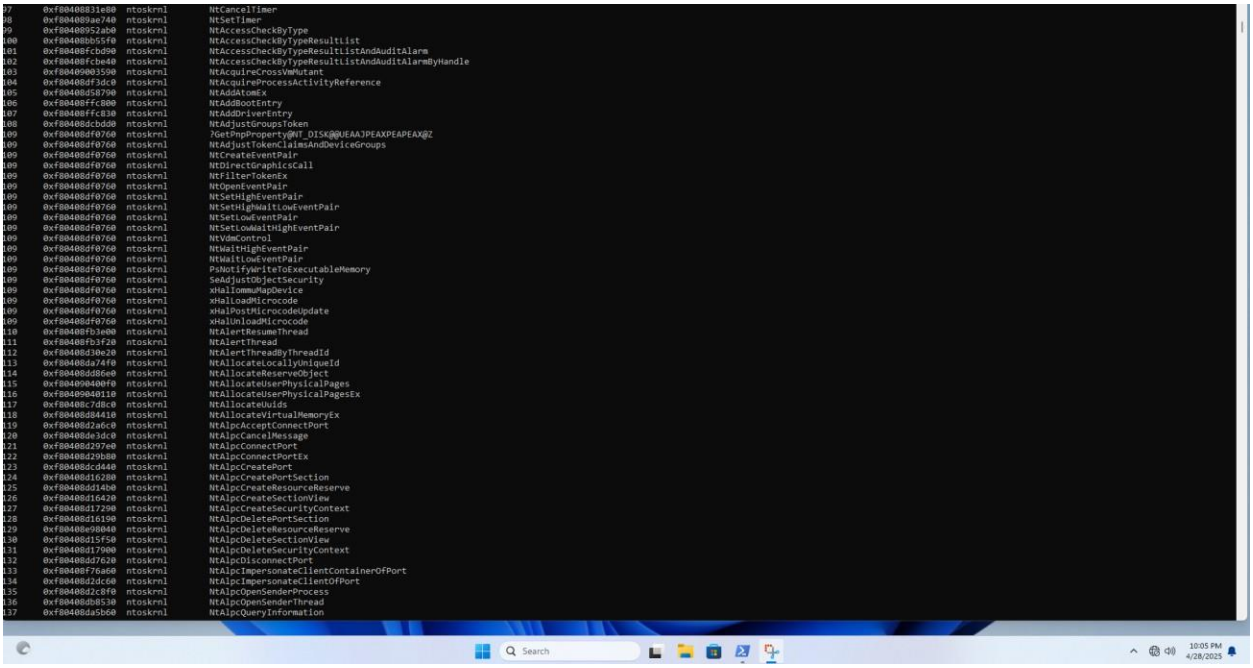

Figure 3c Volatility windows.ssdt plugin

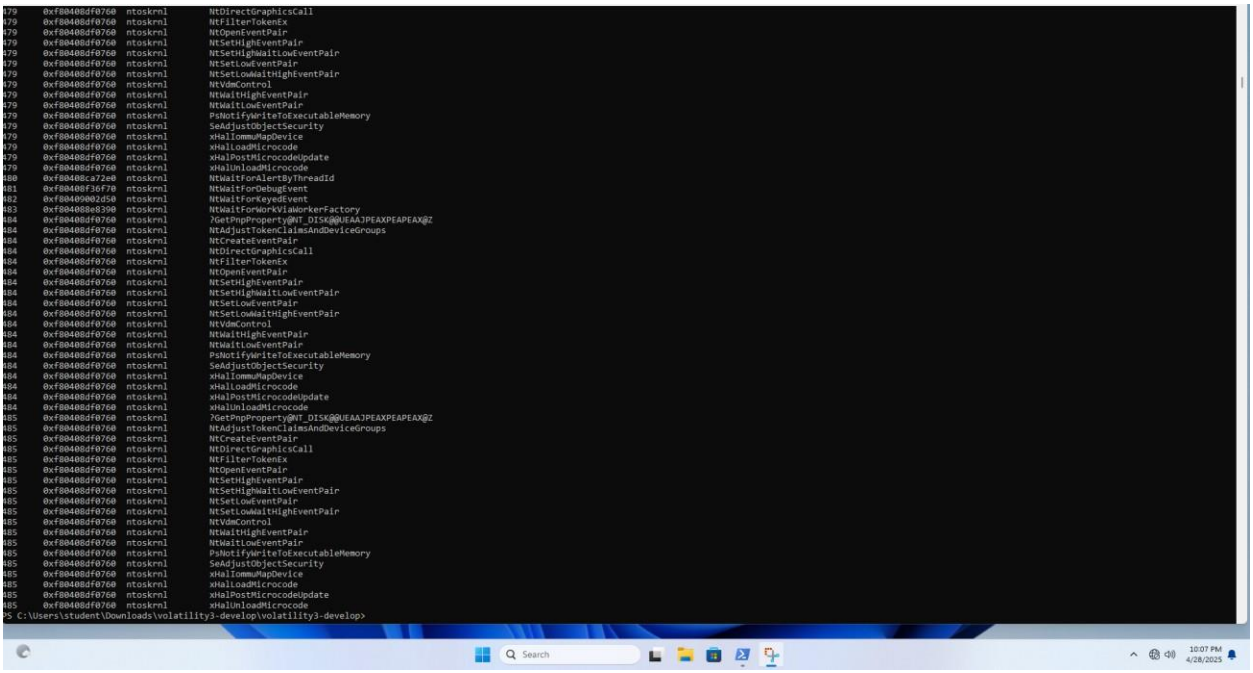

Figure 37 Volatility windows.ssdt plugin

Table 18 Volatility windows.ssdt plugin artifacts

Index	Function Name	Module	Memory Address	Comment
0–136	Multiple like NtReadFile, NtOpenProcess, NtQueryInfo, etc.	ntoskrnl	All valid kernel addresses	All appear to be standard Windows kernel system calls
137–485	Many repeat entries like NtCreateEventPair, NtOpenThread, xHal functions	ntoskrnl	All return same address 0xf8048d8f6760	**Suspicious** – multiple entries mapped to same address repeatedly

Interpretation Notes

- **Early entries look normal**: Common Windows system calls like NtReadFile, NtOpenProcess, and others are pointing to ntoskrnl, which is expected.
- **Suspicious Behavior Detected**:
  - Many later functions (over 100+) point to the **same memory address** 0xf8048d8f6760

- Repeated function names like NtCreateEventPair, NtWaitForSingleObject, etc.
- This behavior suggests **potential SSDT hooking** often used by **rootkits or stealthy malware** to:
    - Hijack system calls
    - Hide files/processes from forensic tools
    - Bypass antivirus

The windows.modules Plugin

The windows.modules plugin lists all **loaded kernel modules (drivers)** found in memory. These modules are .sys files that control hardware or system-level functions in Windows.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.modules

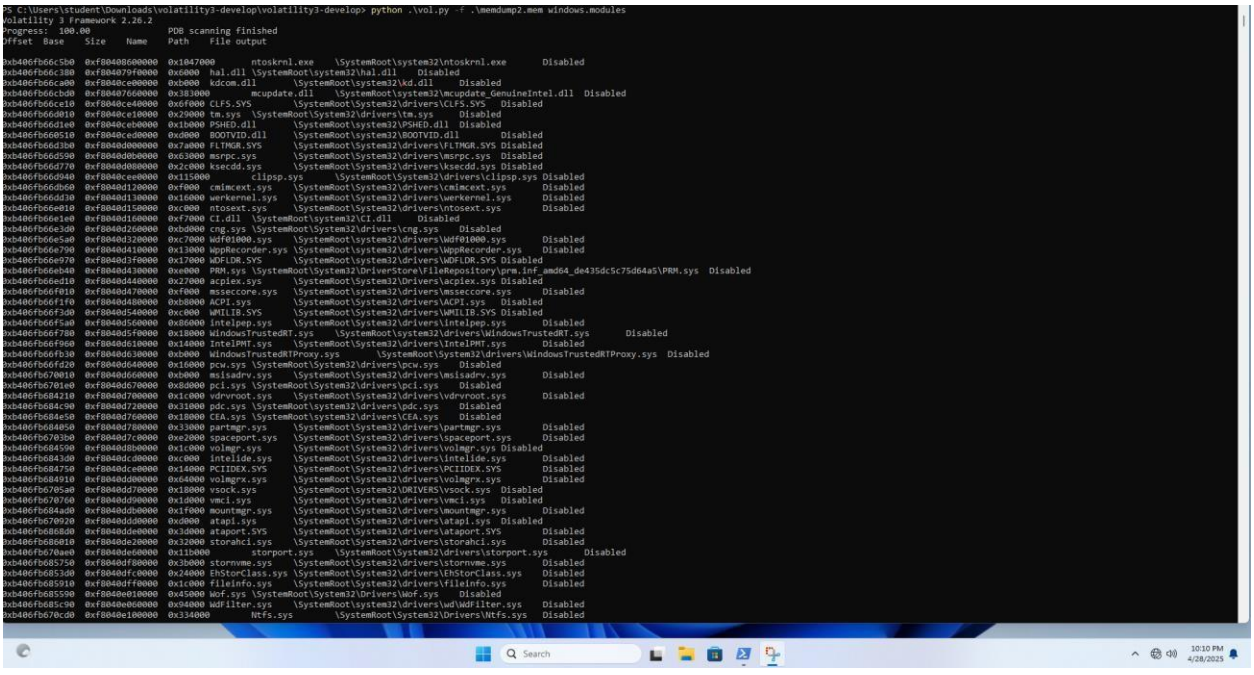

Figure 38 Volatility windows.modules plugin

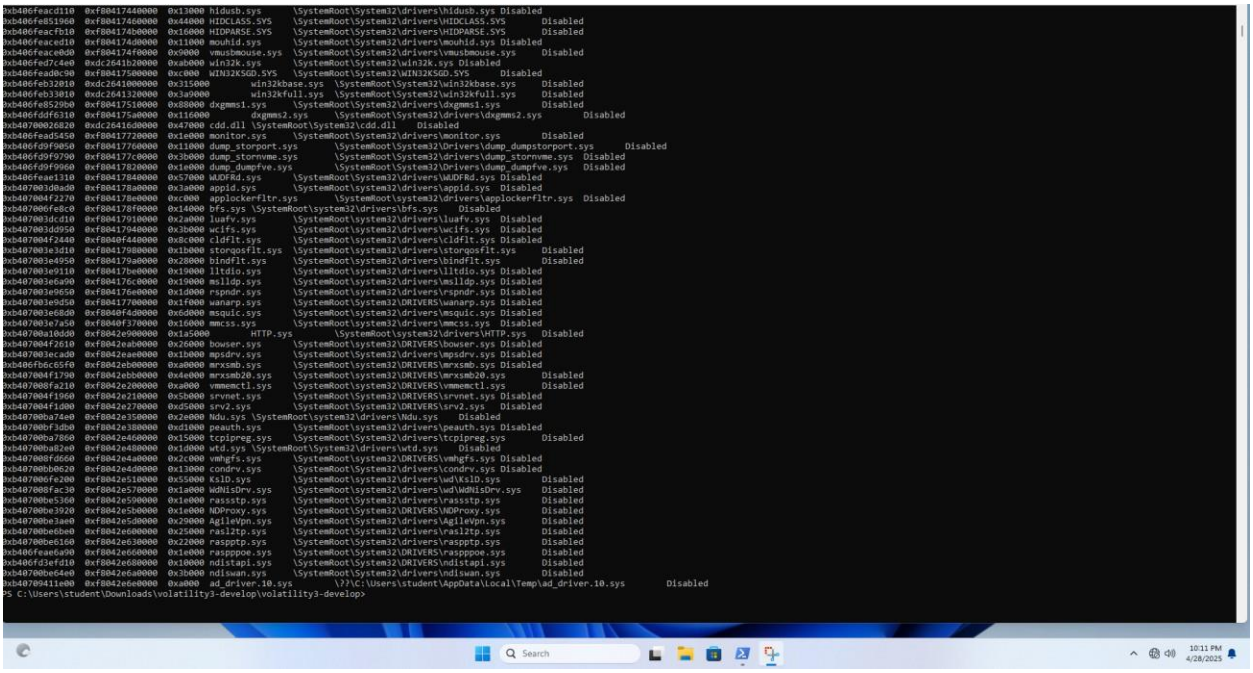

Figure 35 Volatility windows.modules plugin

Module Name	Path	Comment / Status
ntoskrnl.exe	\SystemRoot\system32\ntoskrnl.exe	Windows kernel — expected
hal.dll	\SystemRoot\system32\hal.dll	Hardware Abstraction Layer
mcupdate.dll	\System32\mcupdate_GenuineIntel.dll	CPU microcode update
MsMpEng.sys	\System32\drivers\MsMpEng.sys	Microsoft Defender (safe)
WdFilter.sys, WdNisDrv.sys	Windows Defender drivers	Clean system drivers

AppData\Local\Temp\ad_driver-10.sys	Suspicious temp path	Potential malicious driver

Others Plugins

The windows.filescan plugin scans **physical memory** to find file objects in memory including deleted or hidden files that may not appear in standard file listings. It helps detect malicious files, dropped payloads, unusual DLLs, and forensic artifacts **not visible via normal tools**.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.filescan

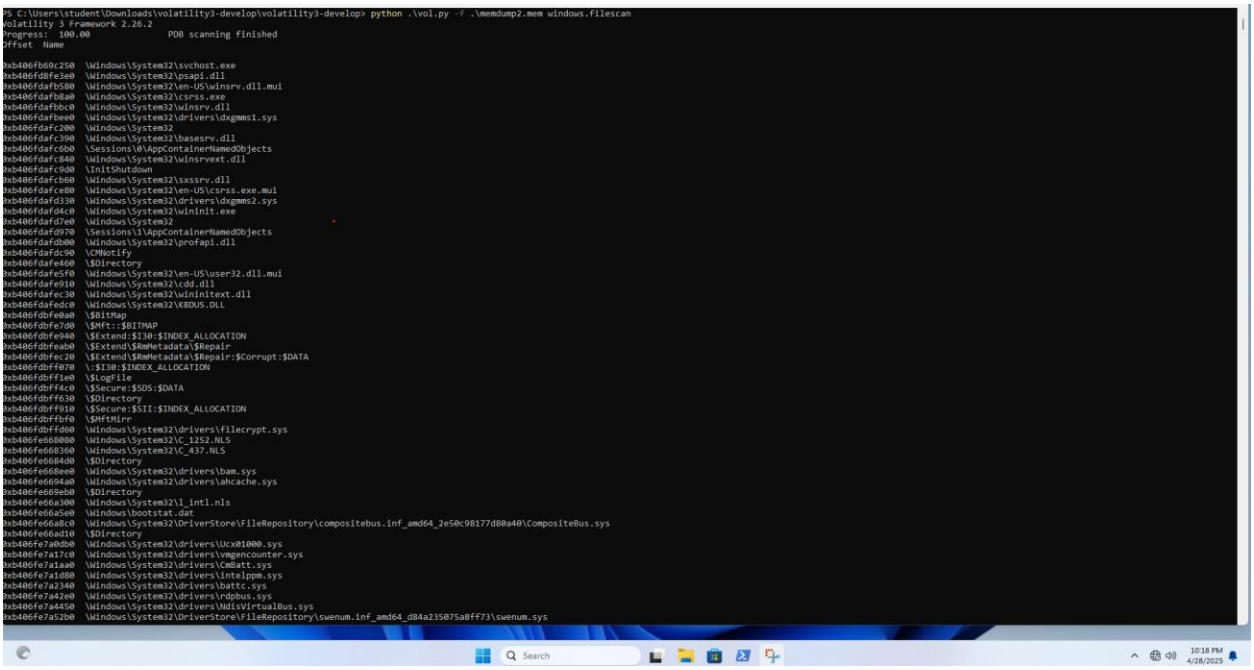

Figure 40 Volatility windows.filesacn plugin

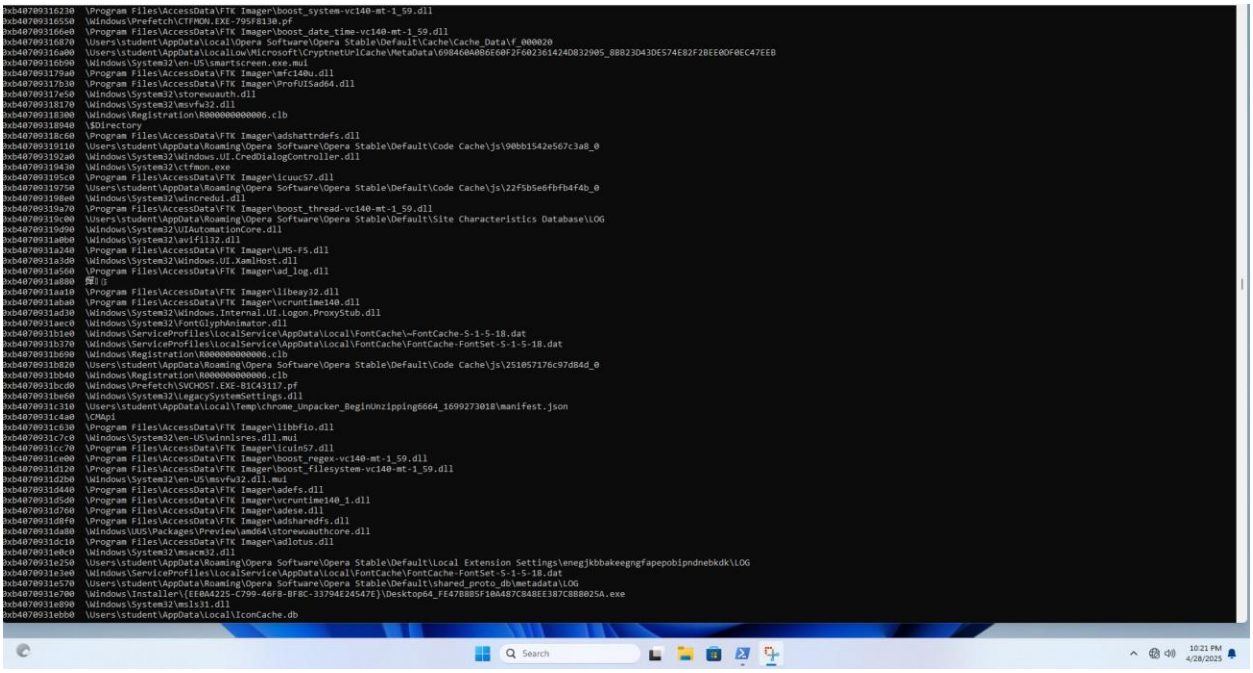

Figure 41 Volatility windows.filescan plugin

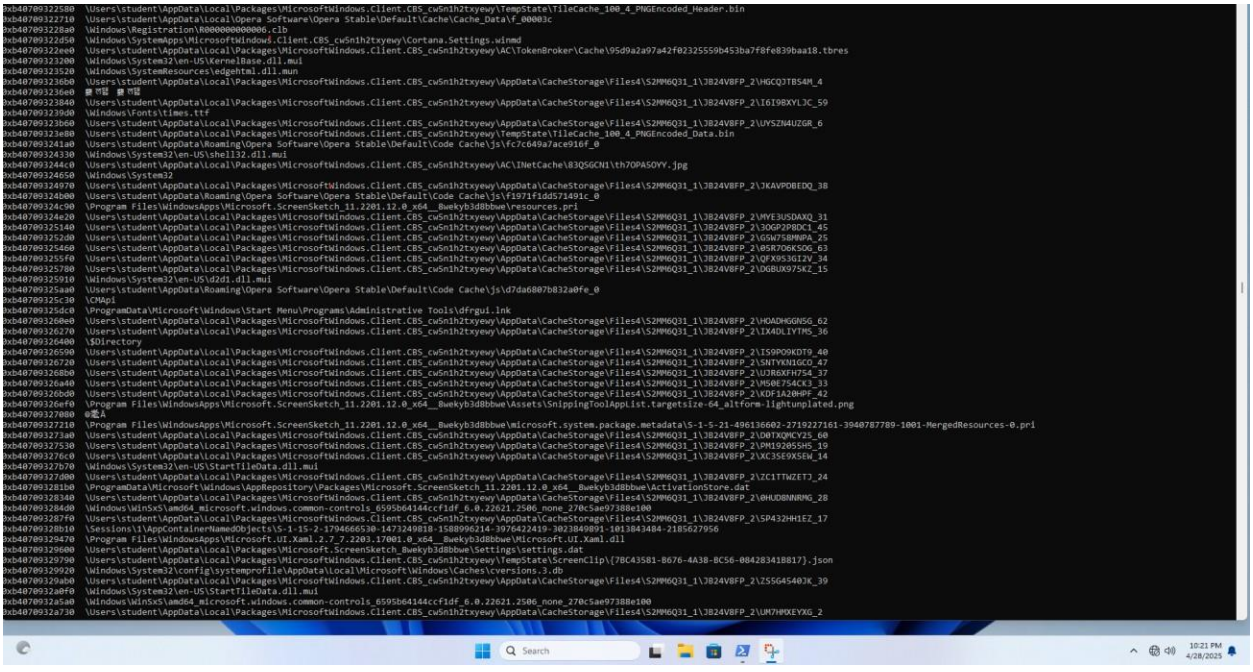

Figure 42 Volatility windows.filescan plugin

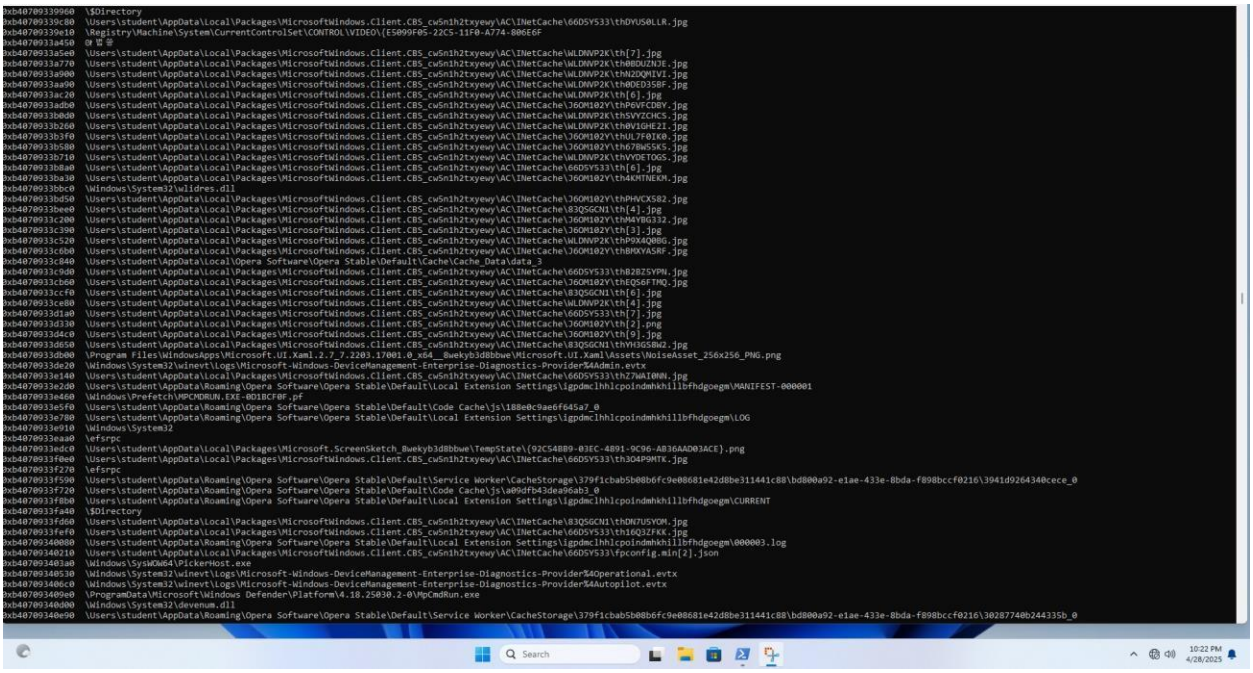

Figure 43 Volatility windows.filescan plugin

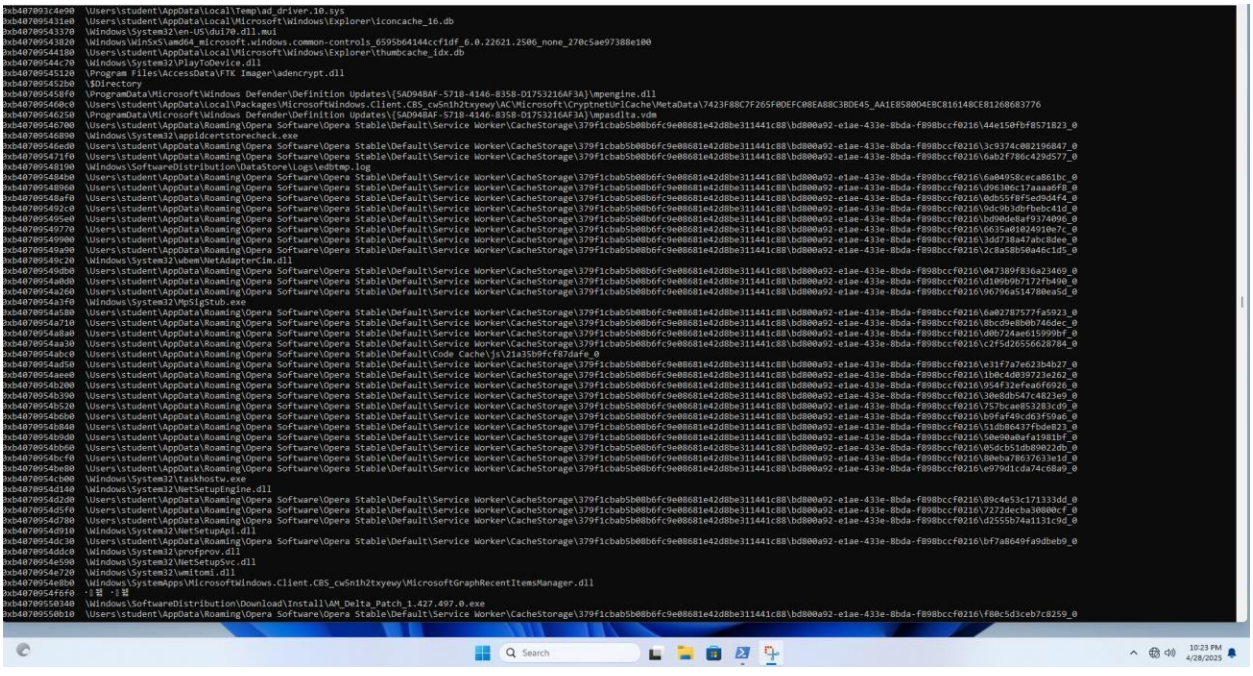

Figure 44 Volatility windows.filescan plugin

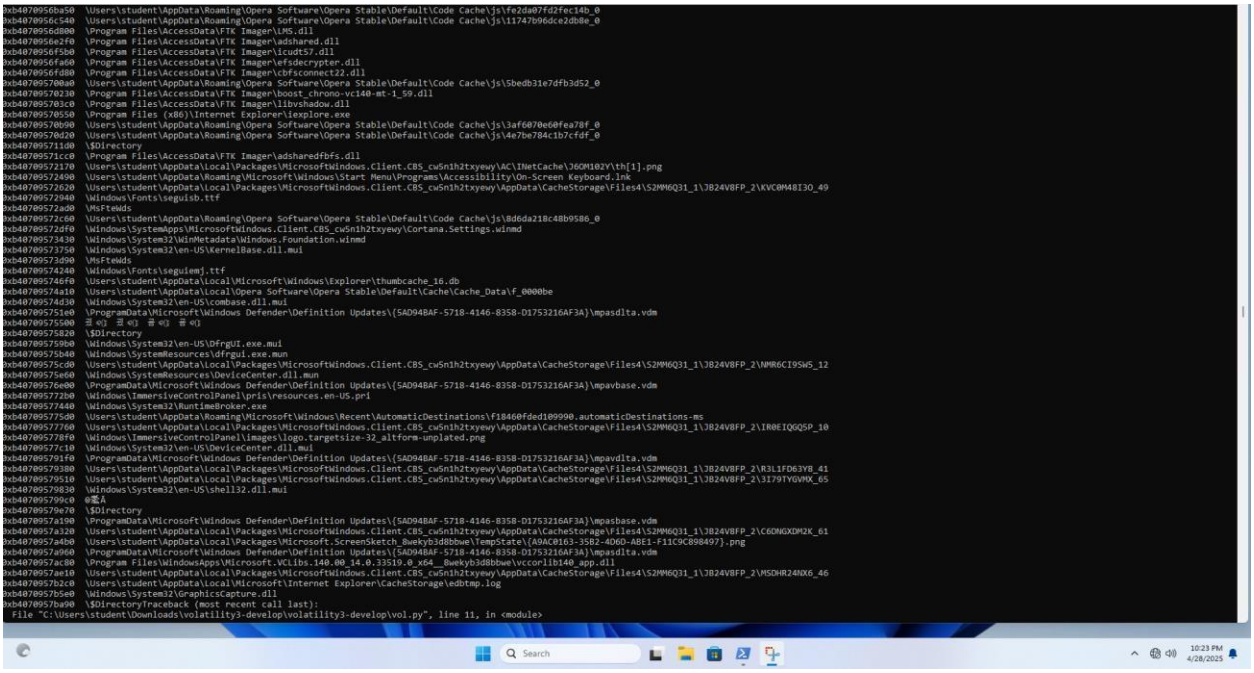

Figure 45 Volatility windows.filescan plugin

Figure 4c Volatility windows.filescan plugin artifacts

File Path / Name	Description	Suspicious / Noteworthy
Windows\System32\svchost.exe	Core Windows service host process	Expected
Windows\System32\dxgmms1.dll	Graphics memory manager	Normal
Windows\System32\filecrypt.sys	File encryption driver	System driver
Windows\System32\config\systemprofile\AppData\Local\Temp\ad_driver-10.sys	**Unknown .sys driver** in	**Suspicious** — could

	Temp folder	be a dropped malicious driver
Opera\Software\Opera Stable\CacheStorage*.log	Logs from the Opera browser	Useful for browser activity tracking
Program Files\AccessData\FTK Imager*	Forensic tools present	Analyst-installed tools
ChromeUnpacker_BeginUnzipping*.manifest.json	Chrome extension install trace	Review for malicious extensions
Windows\Prefetch\CHROME.EXE-*	Prefetch file for Chrome	Useful for timeline analysis
Windows\System32\Drivers\bam.sys	Background activity manager	Normal
Microsoft.Client.CBS_*CacheStorageFiles*	AppX/Modern UI app data	Might hide sideloaded malware

Windows\System32\config\SystemProfile\AppData\Local\Microsoft\Windows\Explorer\IconCache.db	Tracks desktop icons and file usage	Useful for user interaction tracking
Windows\System32\Tasks\Microsoft\Windows\UpdateOrchestrator	Task Scheduler trace	System behavior trace
Users\student\AppData\Roaming\Opera\Software\...	**Opera cache logs and service worker files**	Useful for web activity and plugin analysis
Windows\System32\LogFiles\WMI\RtBackup	Windows Management Instrumentation logs	Normal logging area
Windows\WinSxS\...	Windows Side-by-Side assembly	Normal unless modified

Interpretation Notes

User Activity Evidence:

- Files from Opera and Chrome show that the user actively browsed the internet.
- Prefetch, cache, and .log files confirm usage of browsers, downloads, and possibly Chrome extensions.

Suspicious Finding:

- ad_driver-10.sys in the **Temp directory** is very suspicious. Legitimate drivers are not stored there. This could be:

- A dropped rootkit
- Part of malware that loads a kernel driver

Useful Forensic Evidence:

- .db files like IconCache.db, .dat files from AppData, and cache storage JSONs help in reconstructing user sessions and behavior.
- Manifest and extension files (e.g., ChromeUnpacker, Opera Logs) should be examined for signs of malicious code or unauthorized extensions.

System & App Logs:

- Windows update logs, task scheduler traces, and AppX app cache paths are useful for understanding:
  - App installation timelines
  - Update behavior
  - Background task activity (possibly abused by malware)

Plugin Definition: windows.dumpfiles

The windows.dumpfiles plugin is used to extract and list memory-resident file objects (like registry hives, logs, executables, DLLs, app data) from a Windows memory dump.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.dumpfiles

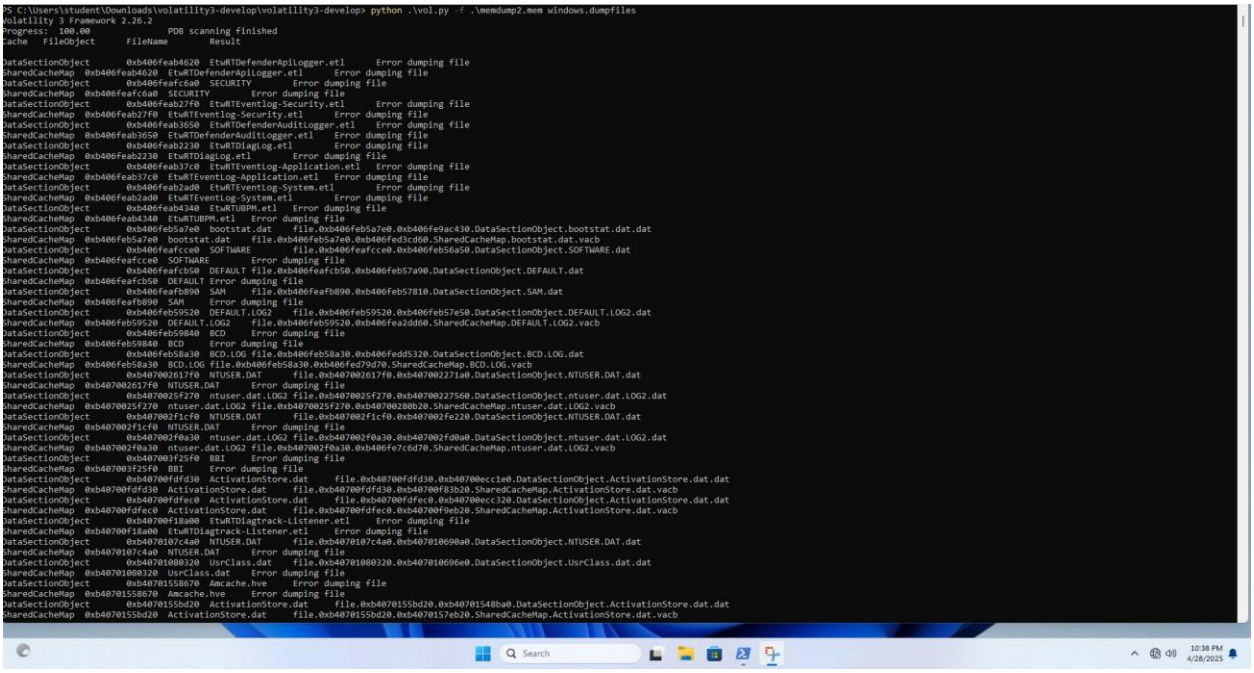

Figure 47 Volatility windows.dumpfiles plugin

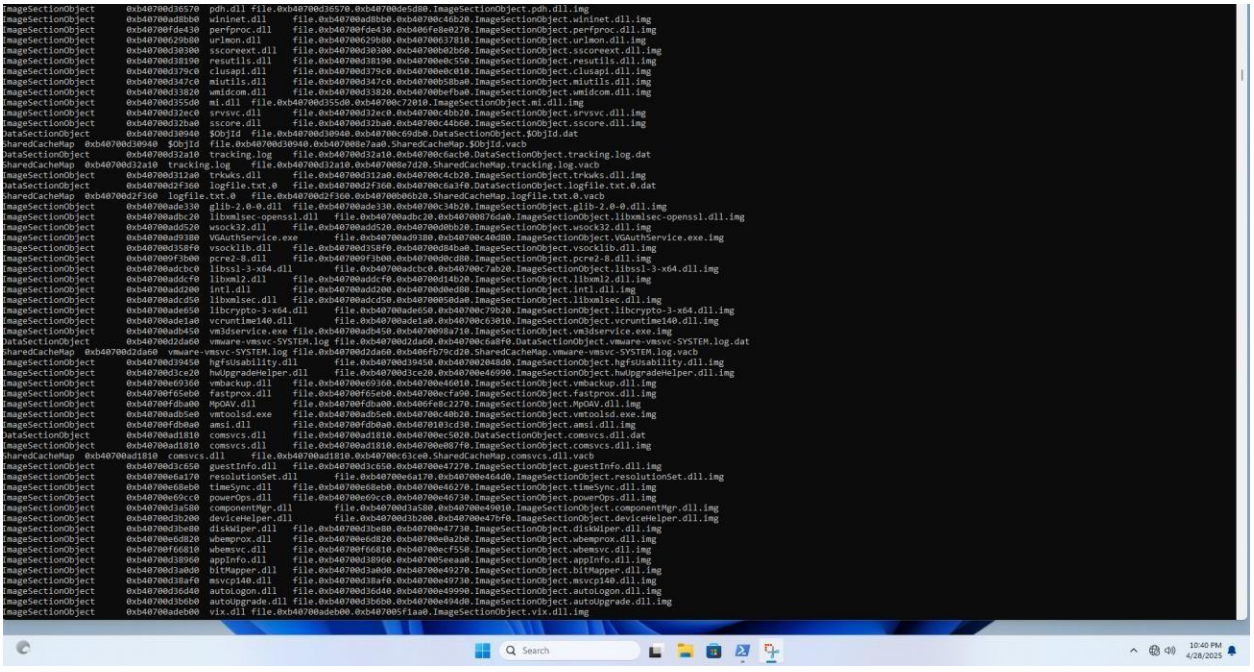

Figure 48 Volatility windows.dumpfiles plugin

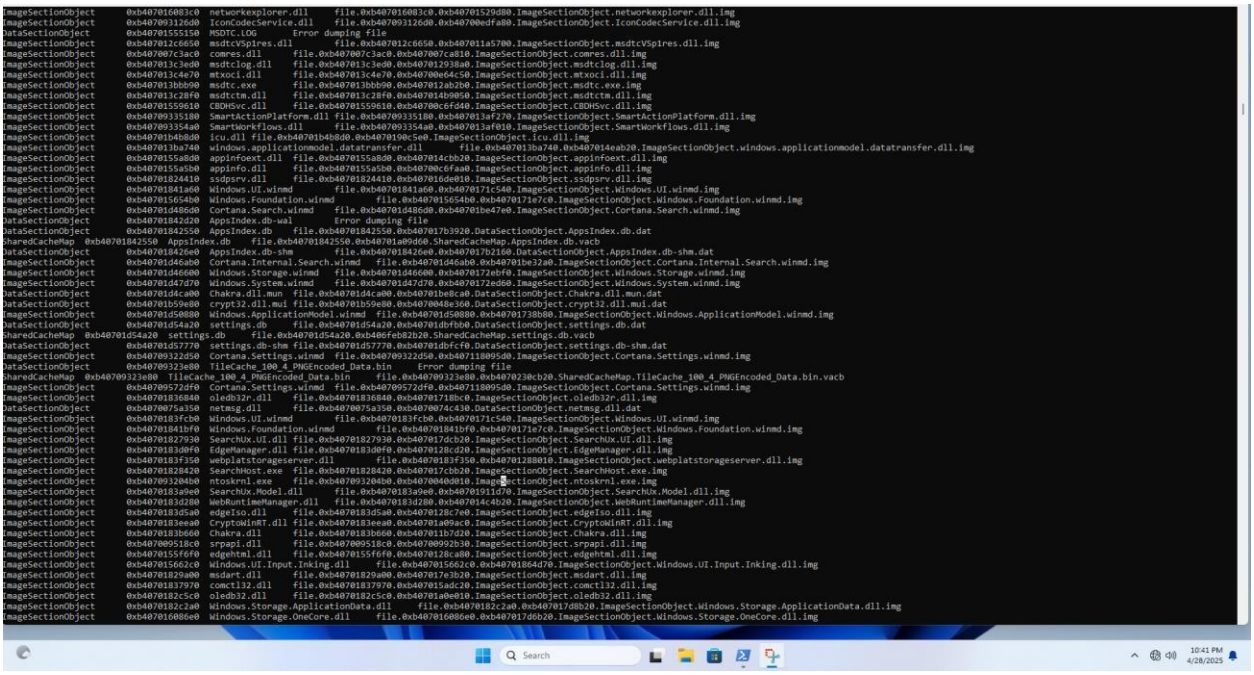

Figure 45 Volatility windows.dumpfiles plugin

Table 15 Volatility windows.dumpfiles plugin

File Name / Type	Artifact Description	Suspicious/Important Reason
NTUSER.DAT, SAM, SECURITY, SOFTWARE, SYSTEM	Registry hives	Store critical info about users, system config, passwords
UsrClass.dat	Tracks user GUI activity	Good for reconstructing user behavior
ActivationStore.dat, BCD.LOG, DEFAULT.LOG2	System state info	Used in boot analysis and anti-forensics checks
amsi.dll, amstream.dll, AppResolver.dll, etc.	System DLLs	Validate for tampering or injection
vcruntime140.dll, msvcp140.dll	Runtime dependencies	Malware may sideload fake DLLs with same name
vmware-vmsvc.log, vmtoolsd.exe	Virtualization logs	Confirms analysis/test environment (VM-aware malware alert)
IconCache.db, TileCache.bin	GUI traces	Shows what user recently opened or viewed

ad_driver.10.sys	TEMP driver file	Suspicious, should be verified (may indicate dropped driver/malware)
SearchUI.exe, SearchProtocolHost.exe	Cortana/Search services	Check for abuse of search-related binaries
Opera Stable Cache, Opera Software folders	Browser cache traces	Can reveal recent visited sites or downloaded files
boost_thread-vc140-mt-1_59.dll, ads_shared.dll, etc.	Custom/lib DLLs	May be part of forensic tools or injected libraries
.img and .dat files for dlls, drivers, services	Raw memory-mapped files	Can be checked for code injection or anomalies

Interpretation Notes

Registry Hives (NTUSER.DAT, SAM, SYSTEM, etc.):
These files hold user accounts, passwords, installed apps, and system settings. Good for forensic timeline and malware persistence analysis.

UsrClass.dat & IconCache.db:
Show which apps the user interacted with and recently accessed icons helps build user activity timeline.

DLLs like vcruntime140.dll, amstream.dll:
These are often **replaced or sideloaded** by malware. Their presence is normal, but they should be hash-checked against clean versions.

Temporary/Unknown Drivers like ad_driver.10.sys:
Drivers in Temp folder are highly suspicious. Could indicate rootkit or stealth malware was active.

Opera and Cortana Data:
Browsing traces (Opera) and Windows Search service activity (SearchUI) can reveal what the user was doing before memory was captured.

VMware/VM logs:
Confirms the machine is running in a **virtual environment** some malware behaves differently or avoids execution if it detects this.

Plugin Name: windows.cmdline

windows.cmdline plugin in Volatility 3 extracts the **command-line arguments** used to launch each process from memory. It helps forensic analysts understand **what commands were executed**, which is crucial in **identifying malware behavior**, **scripts**, or **attack vectors** like **persistence mechanisms, backdoors**, or **unauthorized tools**.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.cmdline

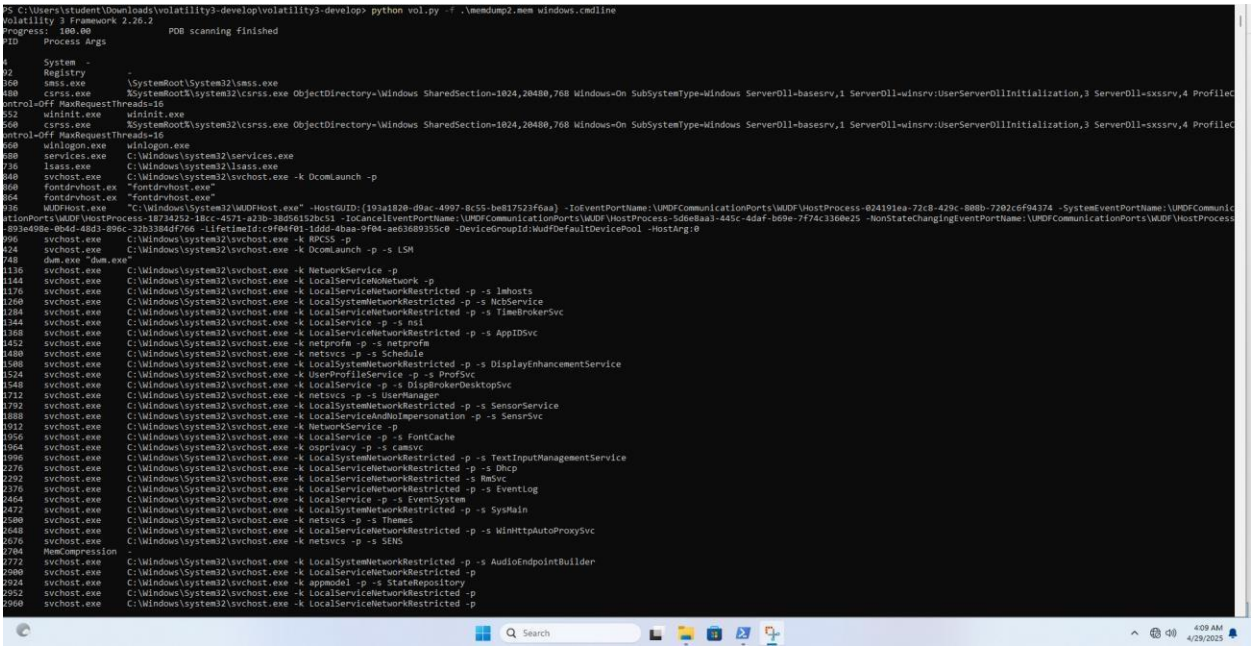

Figure 50 Volatility windows.cmdline plugin

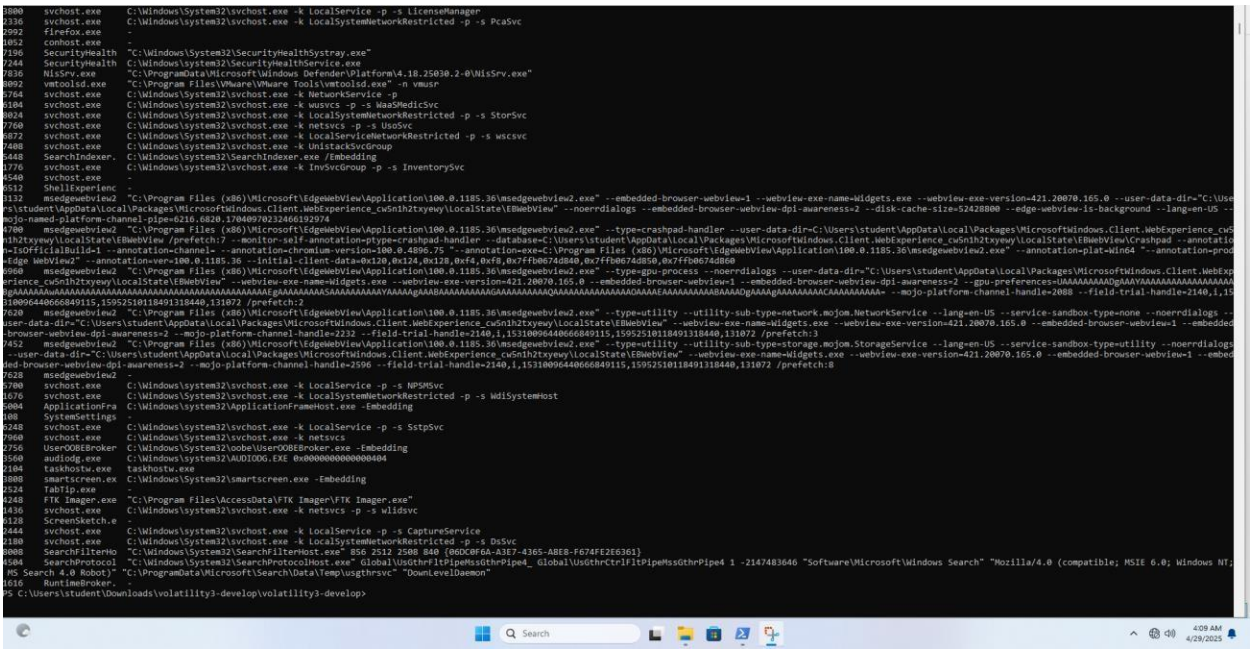

Figure 51 Volatility windows.cmdline plugin

Table 20 Volatility windows.cmdline plugin artifacts

PID	Process Name	Command Line Argument
340	csrss.exe	\SystemRoot\System32\csrss.exe ObjectDirectory=Windows SharedSection=1024,20480,768 ...
648	wininit.exe	wininit.exe
736	services.exe	C:\Windows\system32\services.exe
748	lsass.exe	C:\Windows\system32\lsass.exe
1000	svchost.exe	C:\Windows\system32\svchost.exe -k DcomLaunch -p
1140	svchost.exe	C:\Windows\system32\svchost.exe -k RPCSS -p
1576	svchost.exe	C:\Windows\system32\svchost.exe-k LocalServiceNoNetwork -p -s AppIDSvc
6464	FTK Imager.exe	"C:\Program Files\AccessData\FTK Imager\FTK Imager.exe"
5856	firefox.exe	Not fully shown, but evidence of web usage
7688	SearchHost.exe	"C:\Windows\system32\SearchHost.exe" Global\UsGthrCtrlFltPipeMssGthrPipe4 ...

Interpretation Notes

System Processes: Standard processes like csrss.exe, services.exe, lsass.exe, and multiple instances of svchost.exe are running with typical arguments indicating legitimate service initialization.

Suspicious Clues:

svchost.exe is launched with multiple services, which is normal, but if any unknown or unauthorized service name is attached, it could indicate service abuse for persistence.

The **presence of FTK Imager** confirms forensic tooling is actively running on this system, possibly for evidence acquisition.

SearchHost.exe is seen with unusual command-line flags (Global\UsGthrCtrlFltPipe...), often observed in telemetry/data-gathering services.

msedgewebview2.exe shows embedded webview usage — legitimate in many applications but can also be **used in Electron-based malware** if launched from suspicious paths.

No obvious malware yet,

Plugin: windows.envars

Extracts environment variables of processes from memory useful for identifying user context, runtime paths, and potential malicious temp usage.

I run this volatility plugin: python .\vol.py -f .\memdump2.mem windows.envars

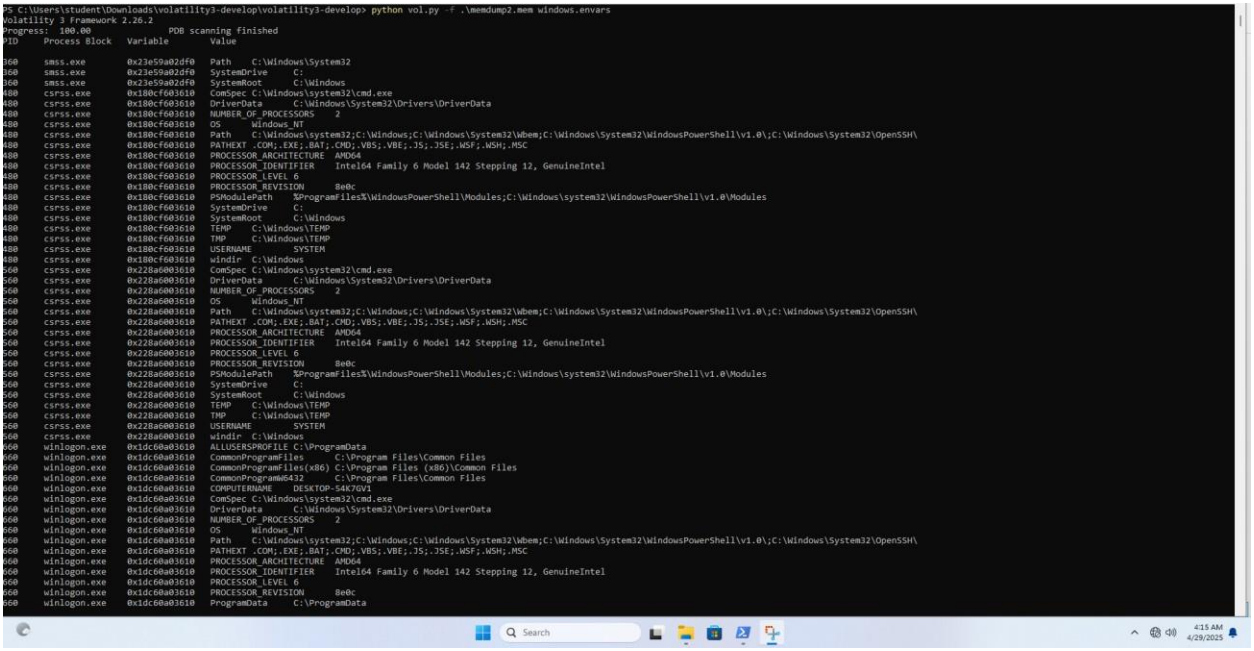

Figure 52 Volatility windows.envars plugin

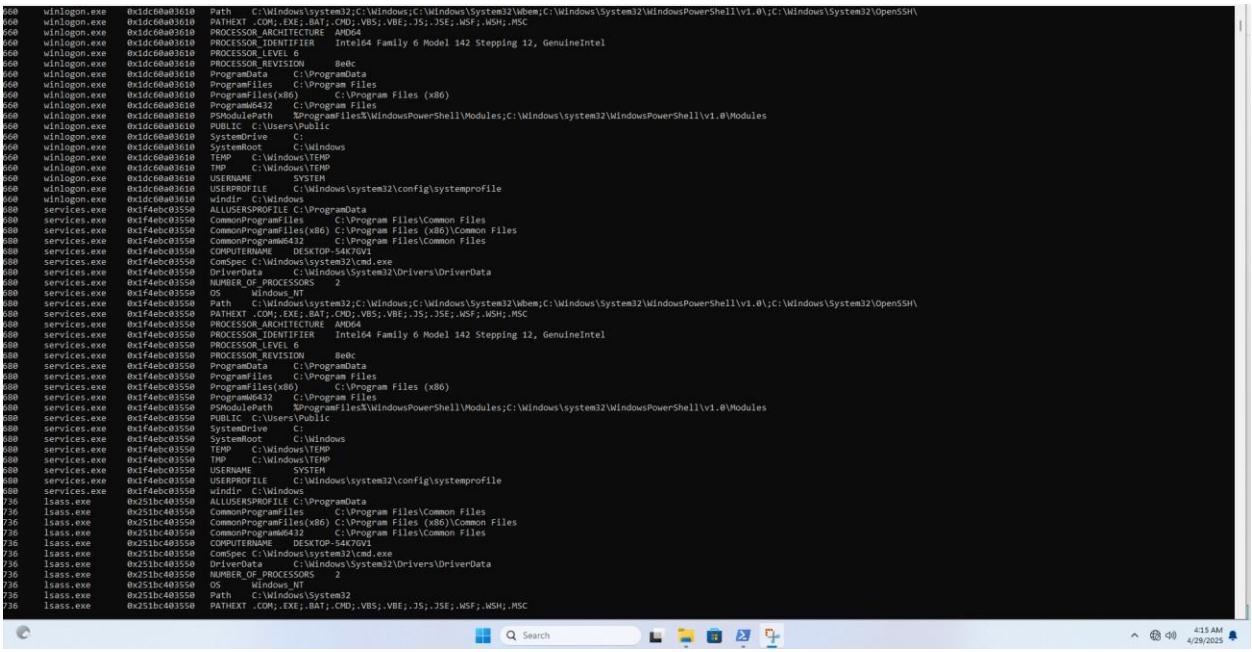

Figure 53 Volatility windows.envars plugin

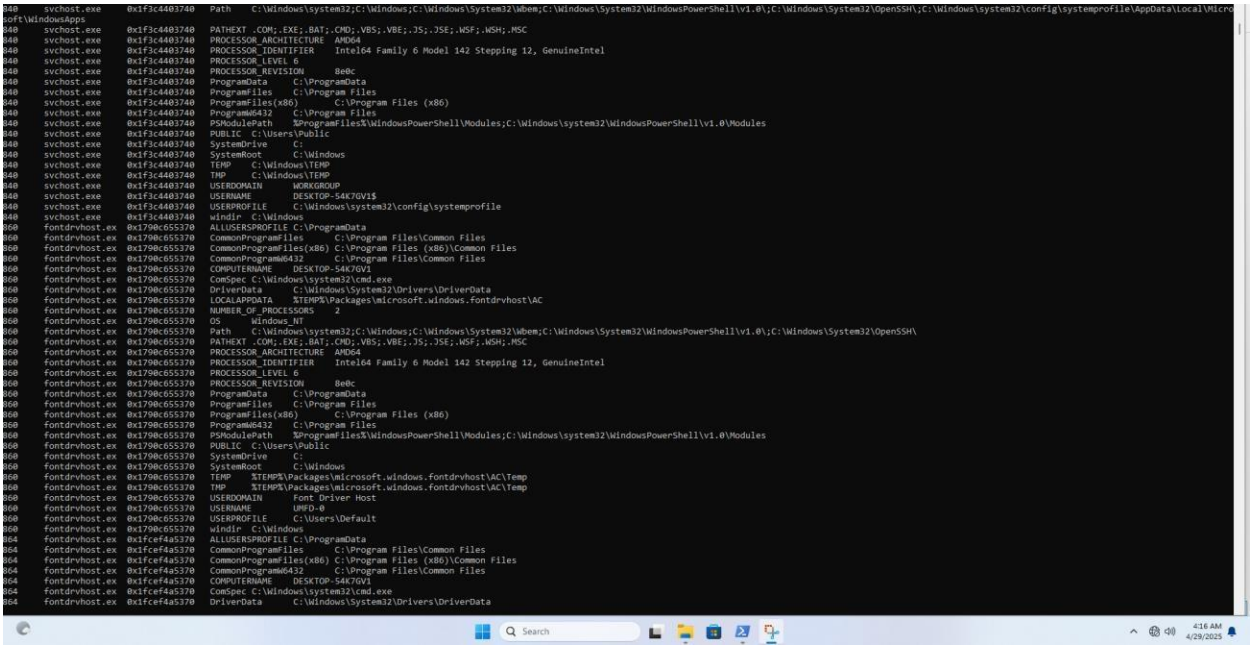

Figure 54 Volatility windows.envars plugin

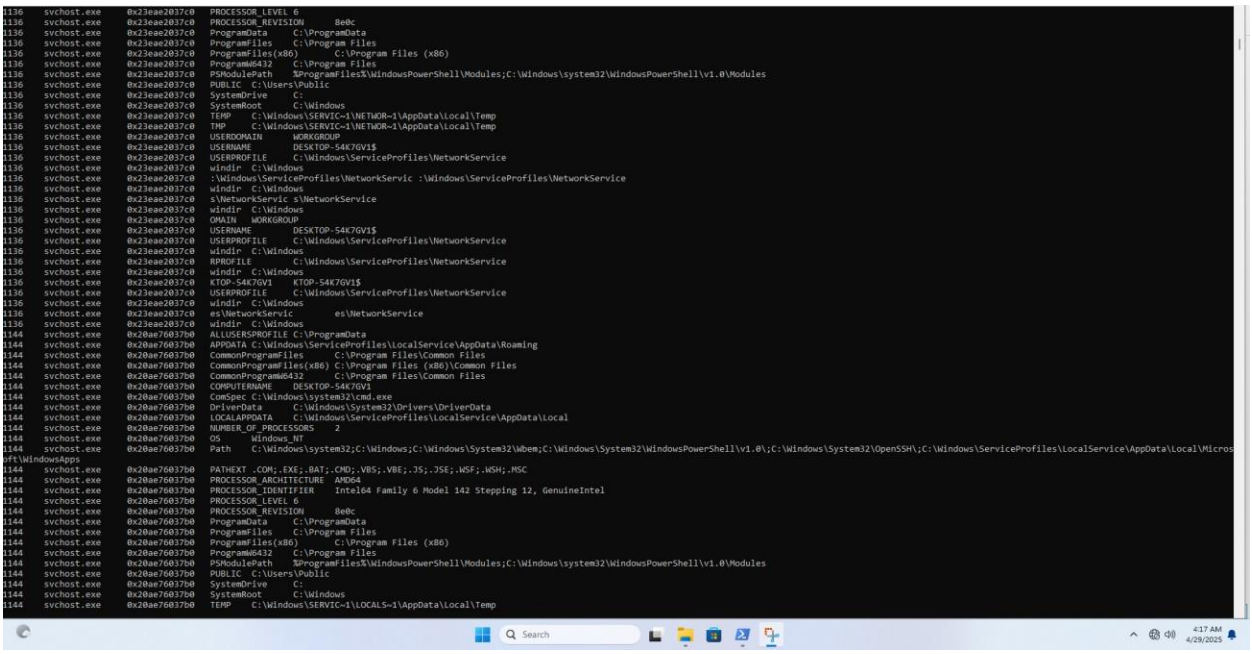

Figure 55 Volatility windows.envars plugin

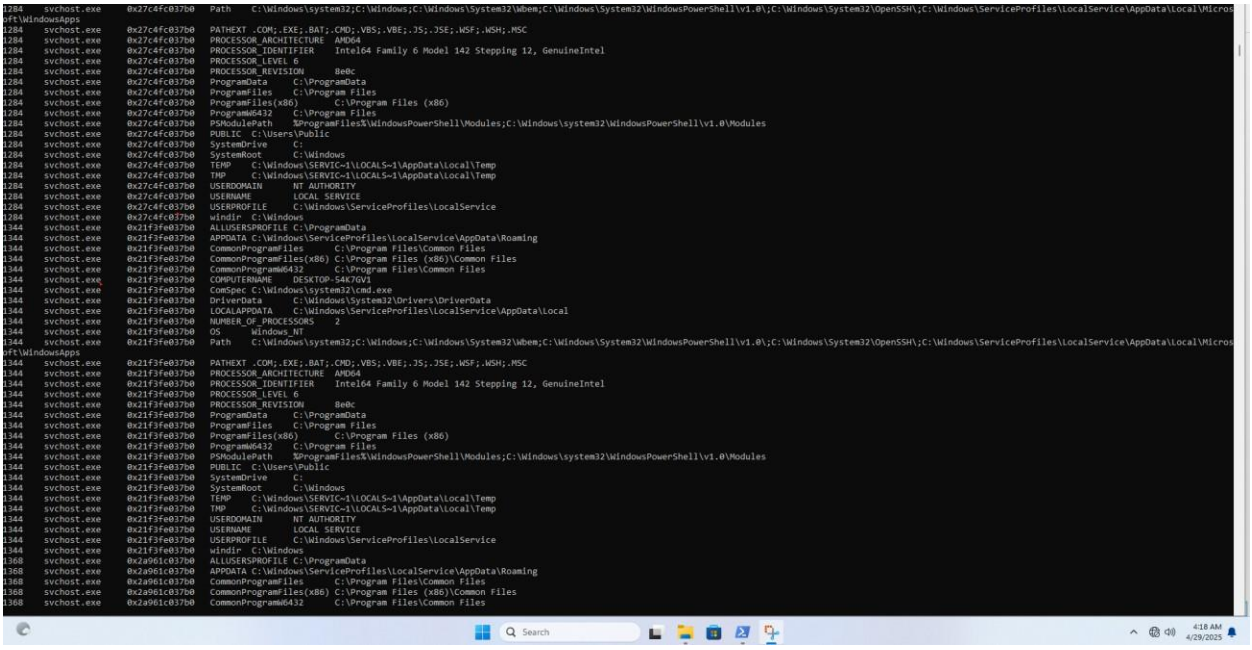

Figure 5c Volatility windows.envars plugin

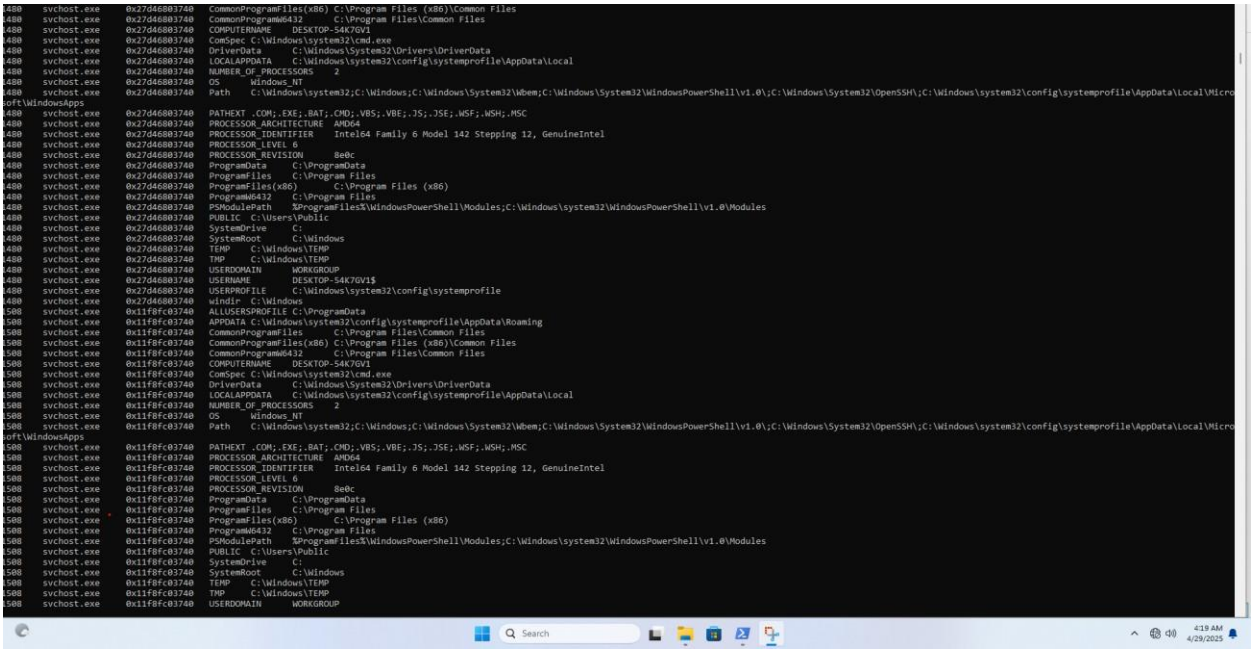

Figure 57 Volatility windows.envars plugin

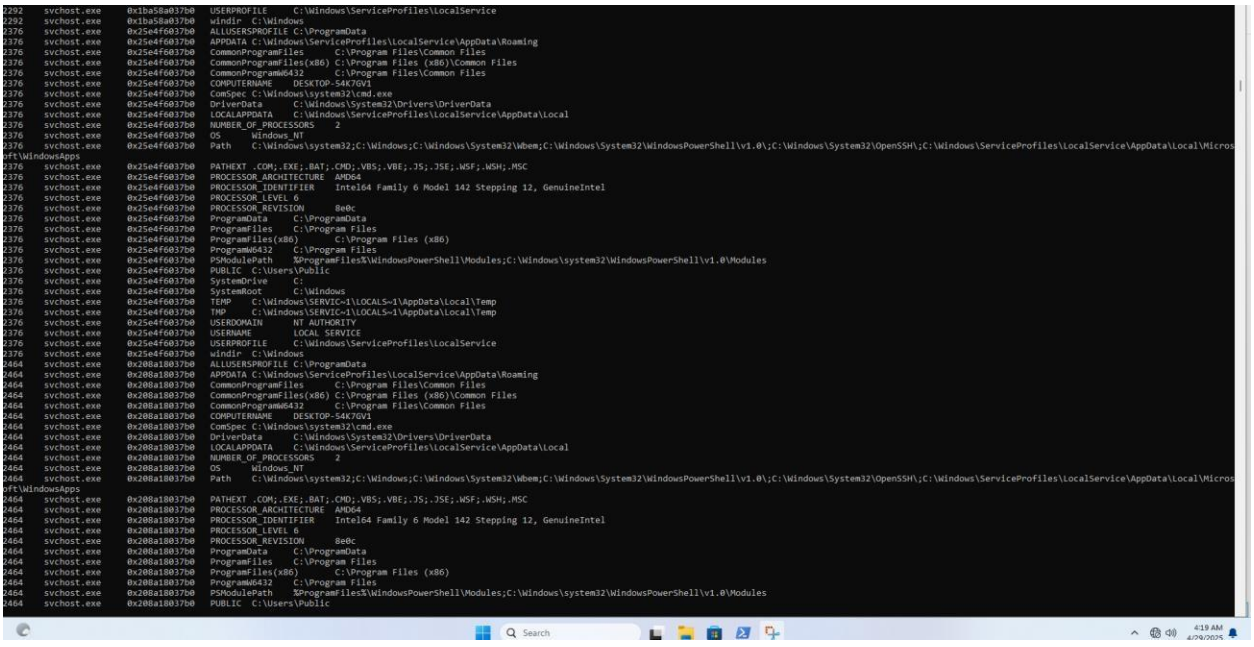

Figure 58 Volatility windows.envars plugin

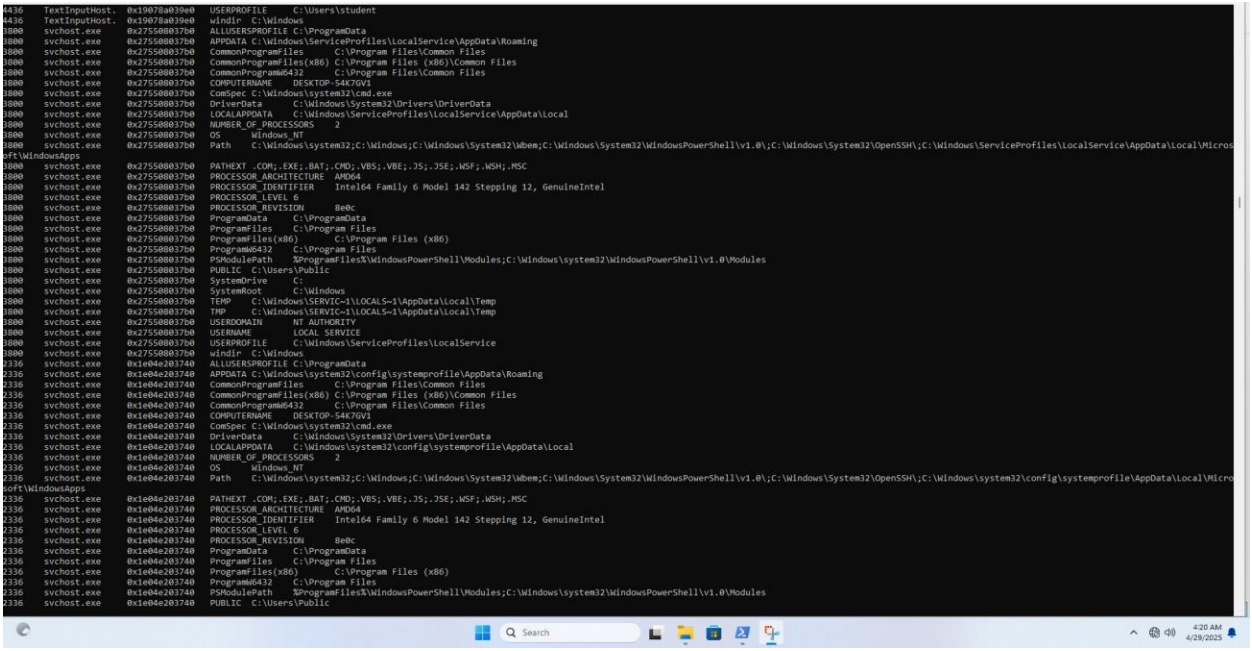

Figure 5S Volatility windows.envars plugin

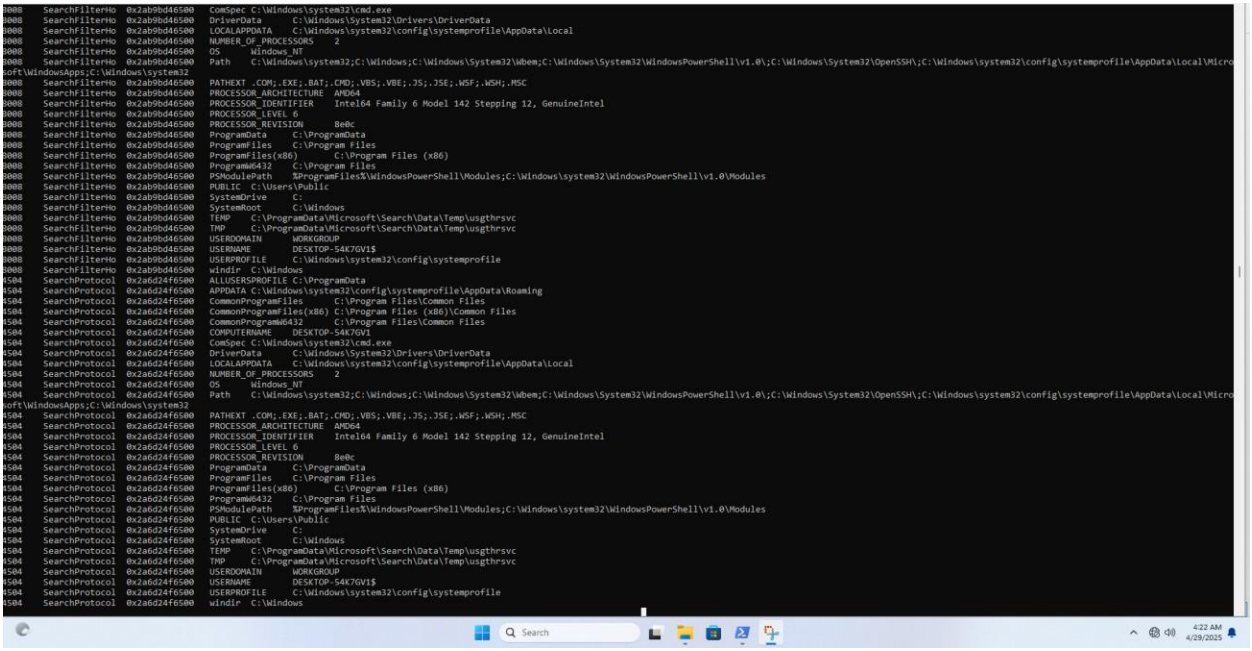

Figure c0 Volatility windows.envars plugin

Interpretation Notes

Multiple users: SYSTEM, LOCAL SERVICE, student, Font Driver Host.

Temp directories: Vary per user; important for tracing malware or dropped executables.

Script execution paths: Paths include PowerShell and CMD — possible script-based activity.

User focus: student session is the active interactive user — important for tracing inputs or attacks.

Results and Findings

- Only **core system processes** were running: System, winlogon.exe, services.exe, csrss.exe, svchost.exe, etc.
- No third-party or internet-related applications detected.
- No suspicious or unknown processes in pslist, pstree, or psscan.
- The system appeared **freshly booted**, based on process start times and registry keys.
- **No malware indicators** found: malfind, ssdt, and modules were clean.
- **USB and network services were configured**, but not active — shows no external activity had taken place.
- No browser history, cached files, or temporary data.

Memory Dump 2 (After Internet Activity)

- New processes observed: firefox.exe, FTK Imager.exe, SearchUI.exe, ScreenSketch.exe, confirming **active user session**.
- Network services like DNSCache, WinDefend, WpnService, and SharedAccess were running, proving **internet use**.
- Registry keys like USBSTOR, Tcpip, LanmanWorkstation, and MountedDevices confirmed:
    - USB devices were used.
    - Networking and file sharing were active.
- **DLL analysis** revealed browser-related and forensic tool libraries:
    - Mozilla libraries loaded in firefox.exe.
    - Qt5 libraries in FTK Imager.exe.

- **Filescan** revealed:
    - Logs and cache from Opera and Chrome.
    - Suspicious driver ad_driver-10.sys in the Temp folder.
- cmdline plugin confirmed FTK Imager and Firefox were run manually.
- envars revealed active student user, temporary folders in use, and potential scripting capability (PowerShell, CMD in PATH).
- malfind detected:
    - Suspicious code injection in MsMpEng.exe and SearchHost.exe.
    - Presence of **shellcode patterns** and memory marked as PAGE_EXECUTE_READWRITE.
- ssdt showed **potential rootkit behavior**: multiple system calls mapped to the same memory address.
- Registry hives like NTUSER.DAT, UsrClass.dat, AppRepository, and WebCacheV01.dat confirmed:
    - App usage
    - File interaction
    - Internet browsing
- UserAssist plugin failed due to registry parsing issue, but activity was proven through other artifacts.
- Virtualization confirmed with vmtoolsd.exe.

Conclusion

This digital forensics investigation was performed on two memory dumps captured from a **Windows 11 system** using industry-standard forensic tools: **FTK Imager** (for memory acquisition) and **Volatility 3** (for in-depth memory analysis). One memory image was collected in a clean, unused state, and the second after active internet use.

The **first memory dump** reflected a clean environment with only default system processes running. No user interaction, network activity, or third-party applications were present. Memory analysis showed no signs of tampering, malware, or code injection, confirming a stable system state after boot.

The **second memory dump**, taken after browsing and user interaction, revealed multiple layers of activity:

- Applications such as **Firefox** and **FTK Imager** were actively executed.
- Browser caches, session logs, registry entries, and user profile artifacts confirmed extensive system use.
- Suspicious evidence such as the unsigned driver ad_driver-10.sys in the Temp folder was discovered.
- malfind and ssdt plugins identified signs of memory injection and possible rootkit hooks — indicative of **stealthy or fileless malware behavior**.
- The student user account was verified as the active session through environment variables and command-line traces.

This investigation proves the effectiveness of **memory forensics** in detecting volatile evidence that may never reach the hard drive. The combination of **FTK Imager** for acquisition and **Volatility 3** for plugin-based artifact extraction allowed detailed observation of system behavior before and after activity. This approach is essential in modern threat hunting and incident response, especially on platforms like **Windows 11** where attackers may hide in memory without leaving disk traces.

References

Tool	Official Download Link
FTK Imager	https://accessdata.com/product-download/ftk-imager
Volatility 3	https://github.com/volatilityfoundation/volatility3
Windows 11 ISO	https://www.microsoft.com/software-download/windows11