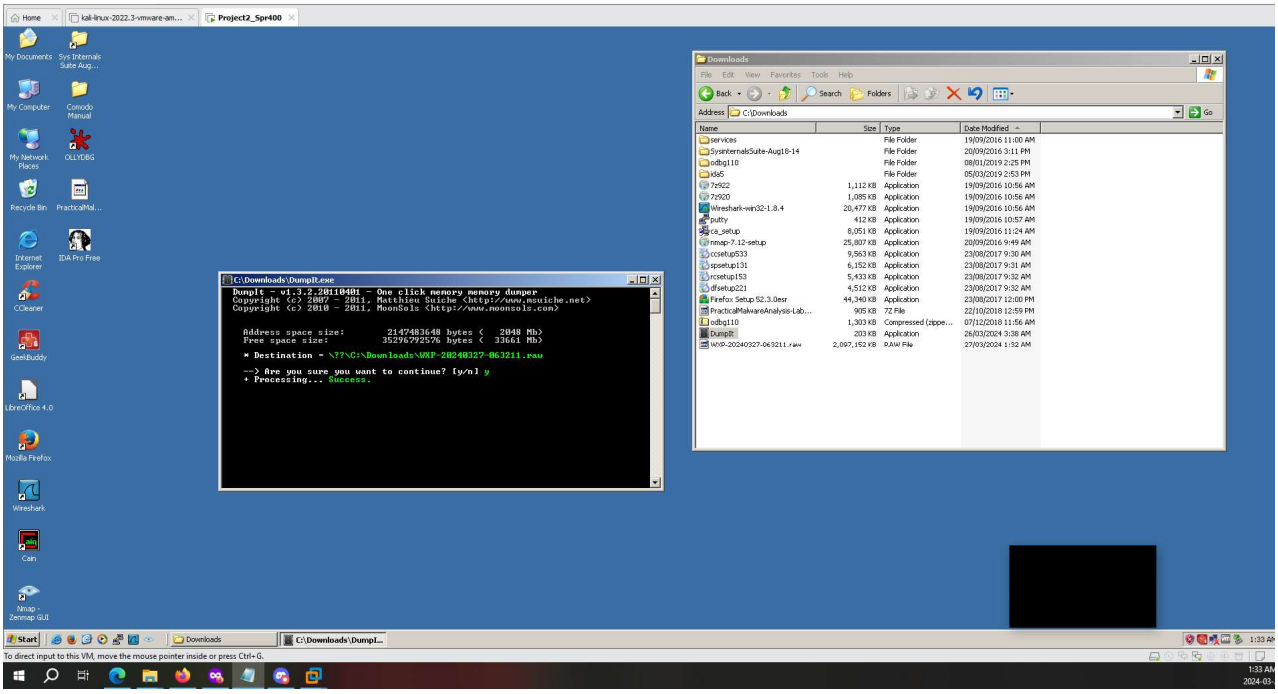
Name: **Abdulfatah Abdillahi**

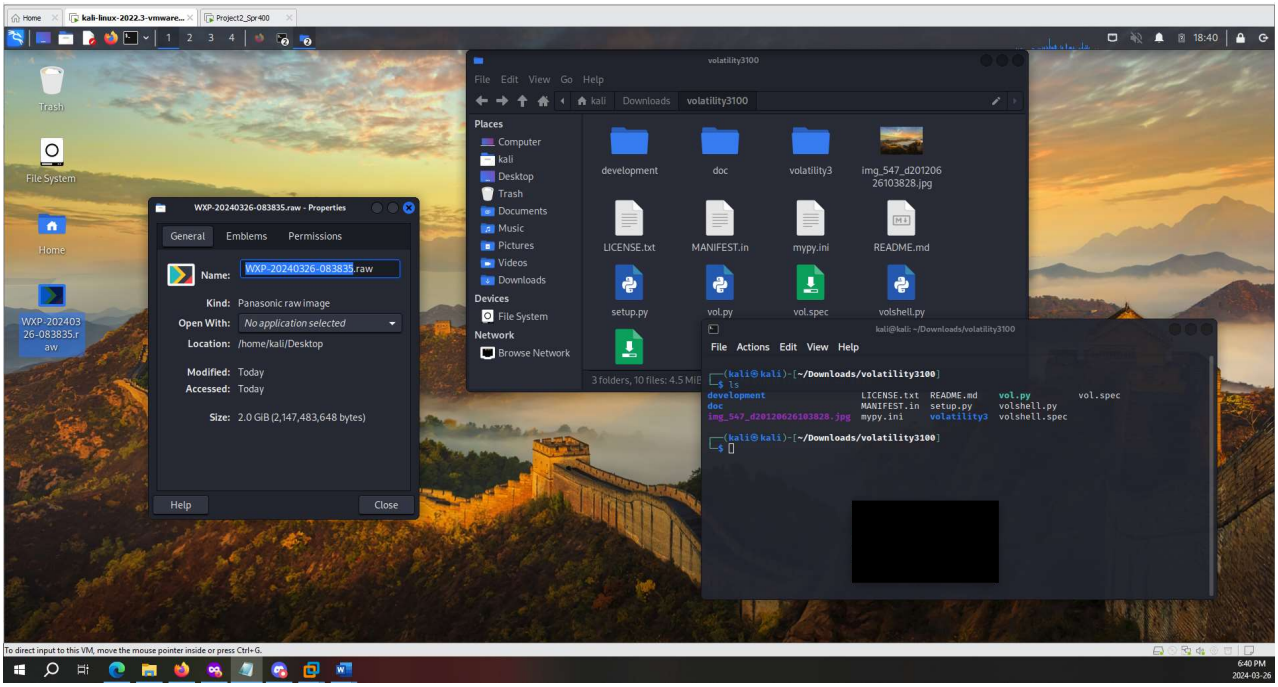
Project: Dumping The Memory

Part 1

Here, you can see me using the dumpit tool after downloading it. Then I used dumpit to dump the memory of my WindowsXP virtual machine. As you can see the memory dump file was saved as WXP-20240327-063211.raw

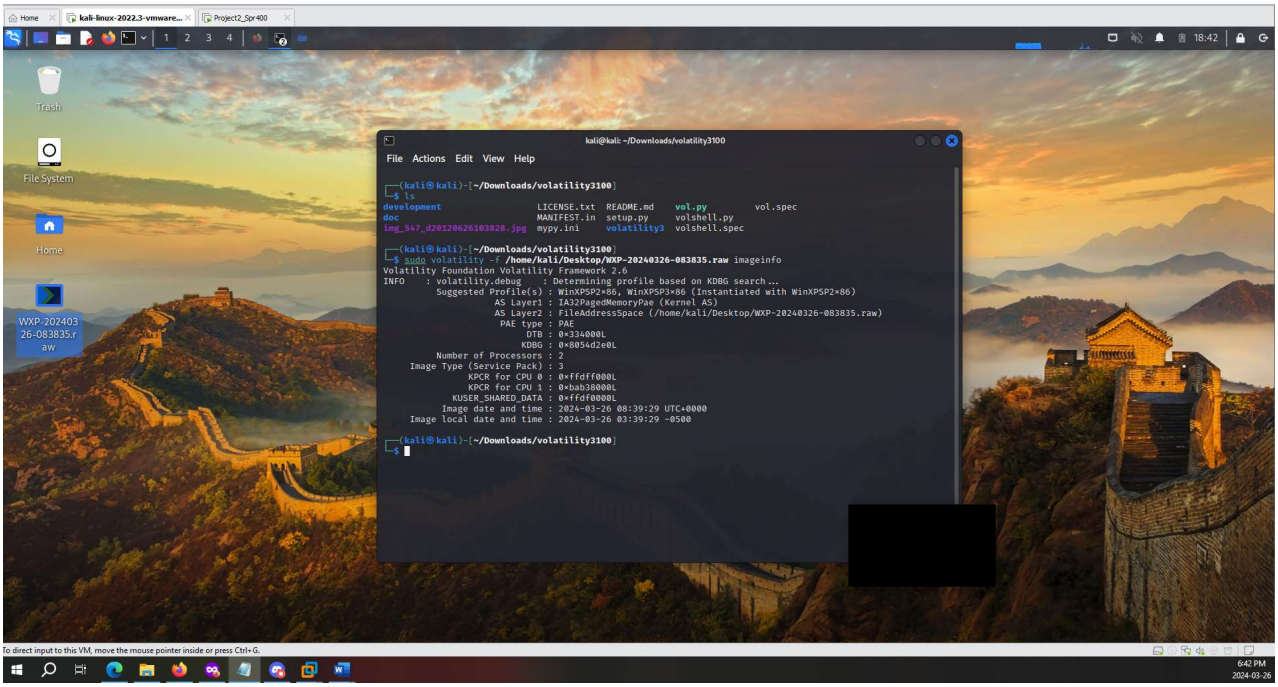


Here, you can see me moving the memory dump to Kali to analyze it using volatility.



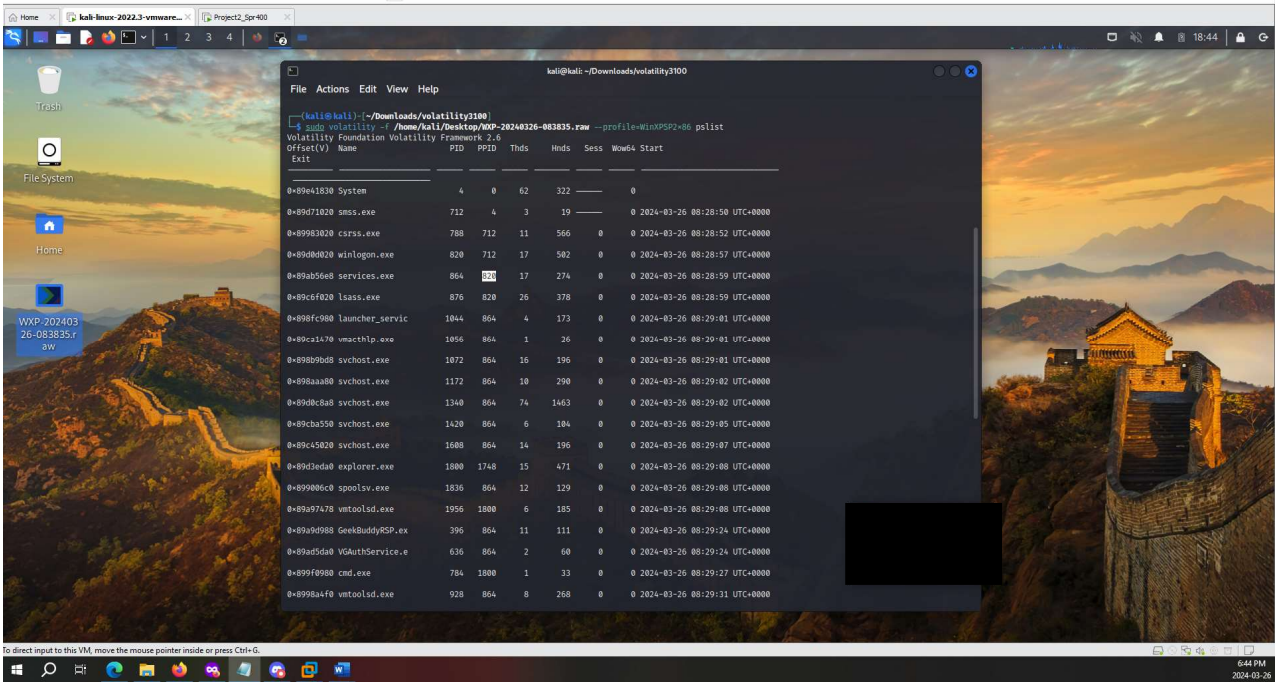
Part 2

Here, you can see me using the **imageinfo** plugin to determine the OS profile of the memory dump. This is necessary because you cannot start the Volatility investigation without knowing the OS profile. Volatility has determined the profile to be WinXPSP2x86.

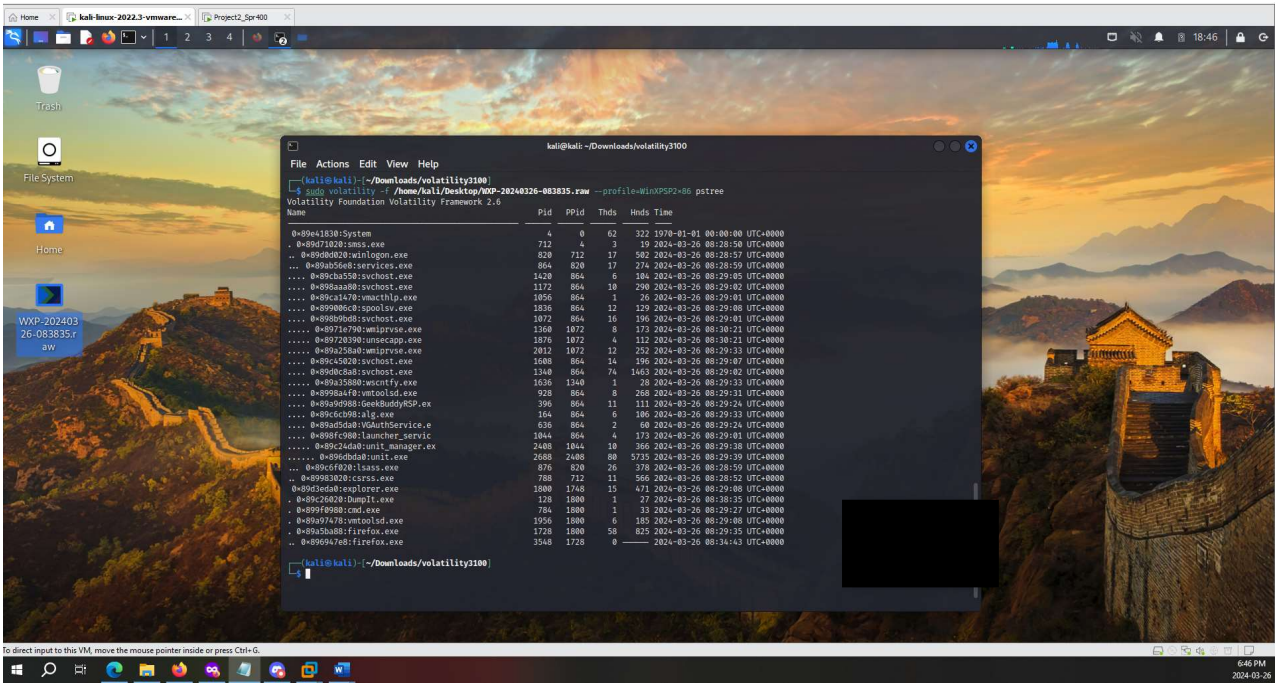


Processes and DLLs

- **Pslist:** This command lists the running processes in the memory dump. Meaning it will give you a list of processes that were running when the memory dump was taken. For example, it shows **firefox.exe** process which is responsible for launching and managing the Firefox web browser on the computer.

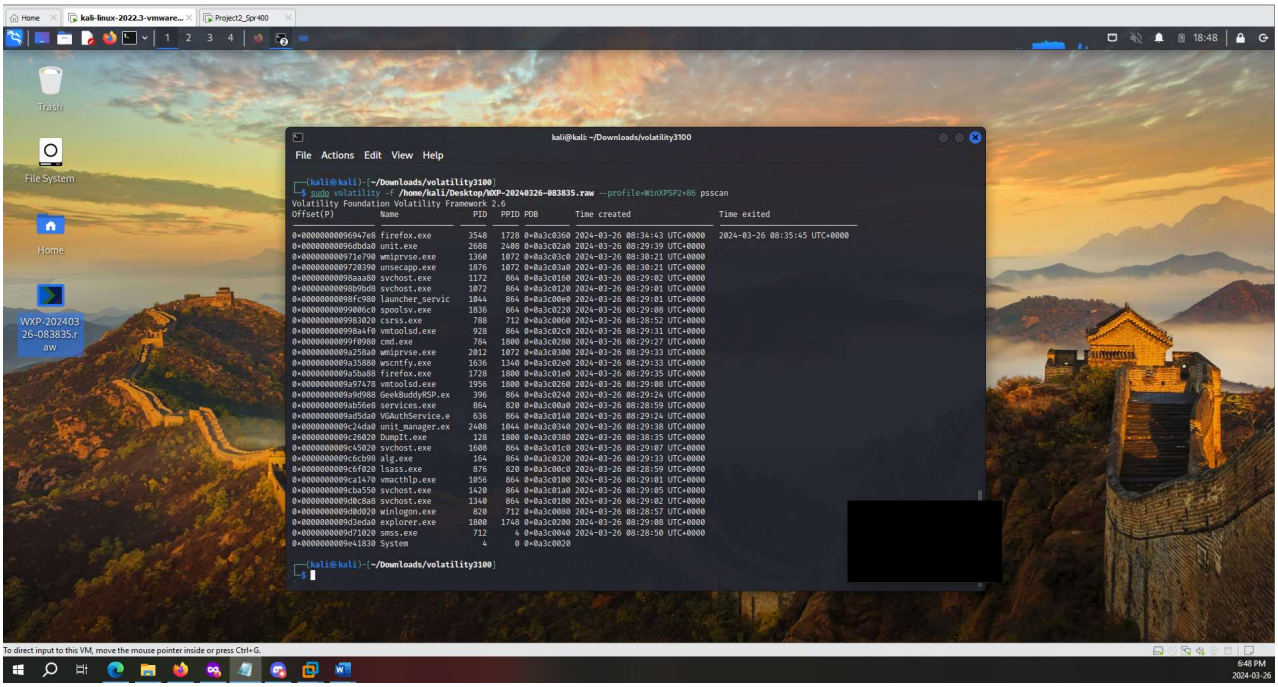


- **Pstree:** This command provides a tree-based view of the running processes. Meaning it will give you a tree view of the processes, showing parent-child relationships. For example, smss.exe (PID 712) is a child of System (PID 4), and winlogon.exe (PID 820) is a child of smss.exe (PID 712). This continues down the tree, showing the hierarchy of processes.



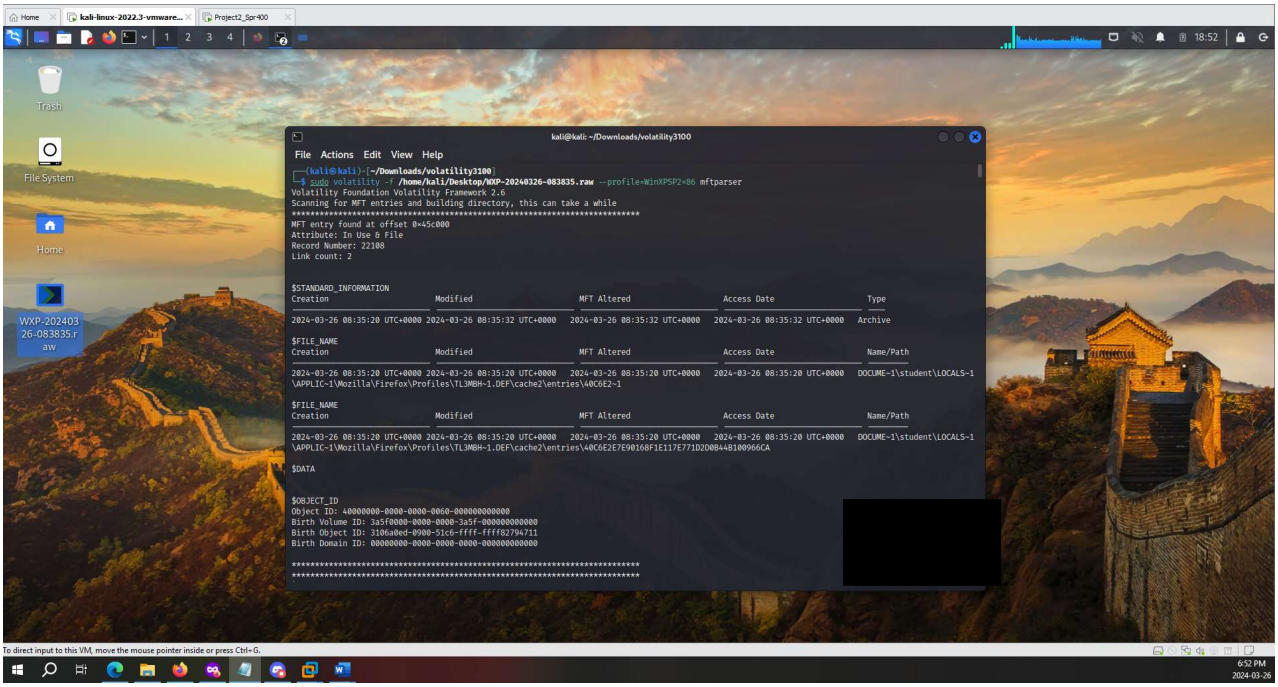
- **Psscan:** This command scans for EPROCESS blocks in memory, which can sometimes find processes that are not in the active process list. This can be useful for finding hidden or

terminated processes. For example, it shows you various system and user processes running, such as System, smss.exe, csrss.exe, etc.



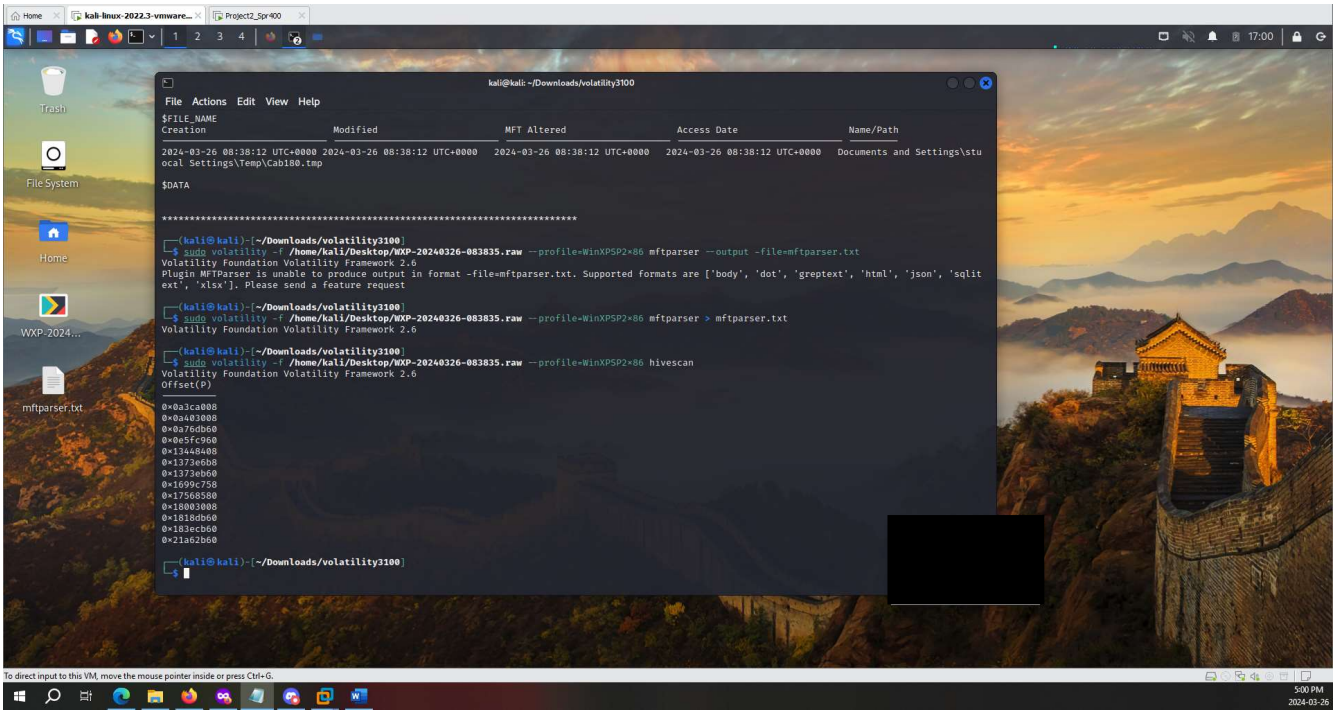
File System

- **mftparser:** This command is used to parse the Master File Table (MFT) from a memory dump. This can provide valuable information about files on the system. For example, it shows several attributes:
  - **$STANDARD_INFORMATION:** This contains standard information about the file, such as its creation time, modification time, MFT altered time, and access time. It also shows the type of the file (e.g., Archive).
  - **$FILE_NAME:** This contains the name of the file and its path. It also shows the same timestamps as in *$STANDARD_INFORMATION*. There can be multiple $FILE_NAME attributes for a single file, representing different names of the file (e.g., short name and long name).
  - **$DATA:** This attribute contains the actual data of the file. In this case, it seems to be empty.
  - **$OBJECT_ID:** This attribute contains the object ID of the file, which is a unique identifier that can be used to track the file even if it's moved to another volume.



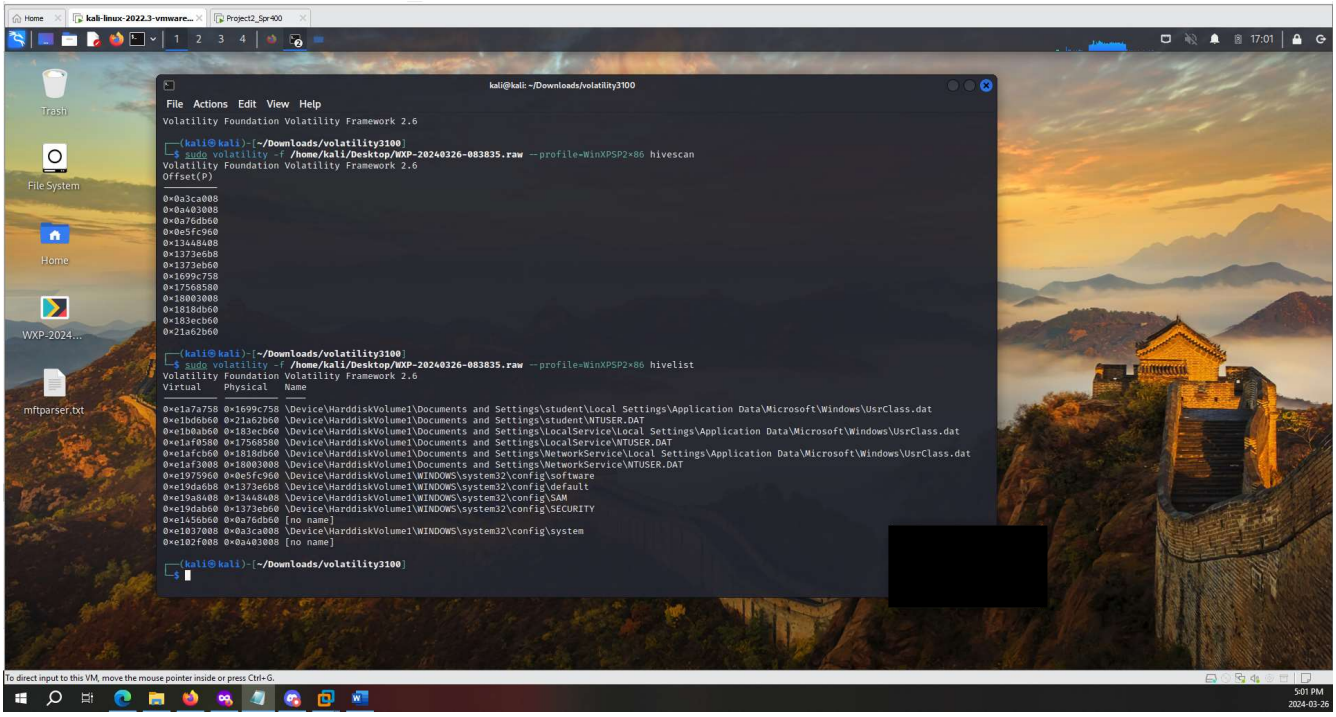
Registry

- **Hivescan:** This command locates the registry hives in memory. Meaning it will give you a list of registry hives found in the memory dump. For example, some common hives include *SAM, SECURITY, SOFTWARE, SYSTEM*, and user profile hives. As you can see below the output shows the offsets of each hive.



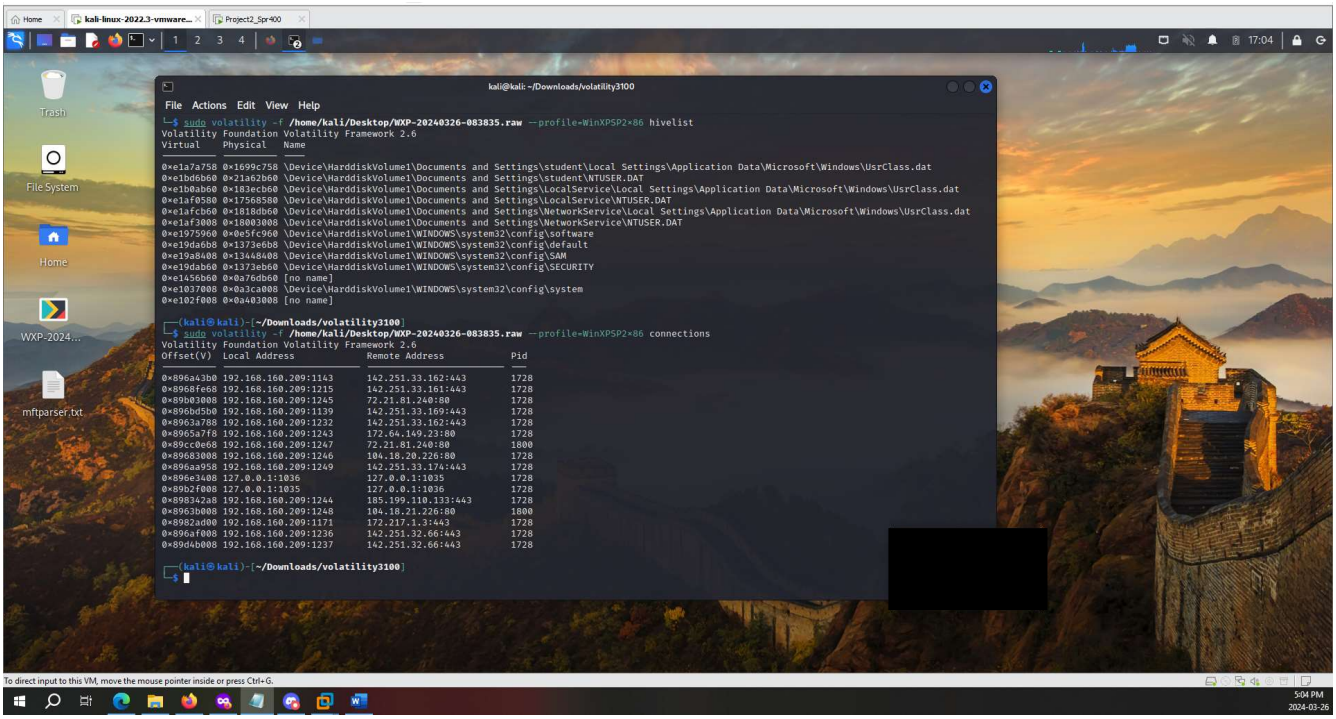
- **Hivelist:** This command lists the virtual addresses of the registry hives in memory. Meaning it will give you a list of registry hives and their locations in memory. From the output, you can see various system and user registry hives (this is normally combined with

hivescan plugin). For example, **\Device\HarddiskVolume1\WINDOWS\system32\config\software** is the *SOFTWARE* hive, which contains software configuration data for the system. **\Device\HarddiskVolume1\Documents and Settings\student\NTUSER.DAT** is the *NTUSER.DAT* hive for the student user, which contains user-specific registry data.

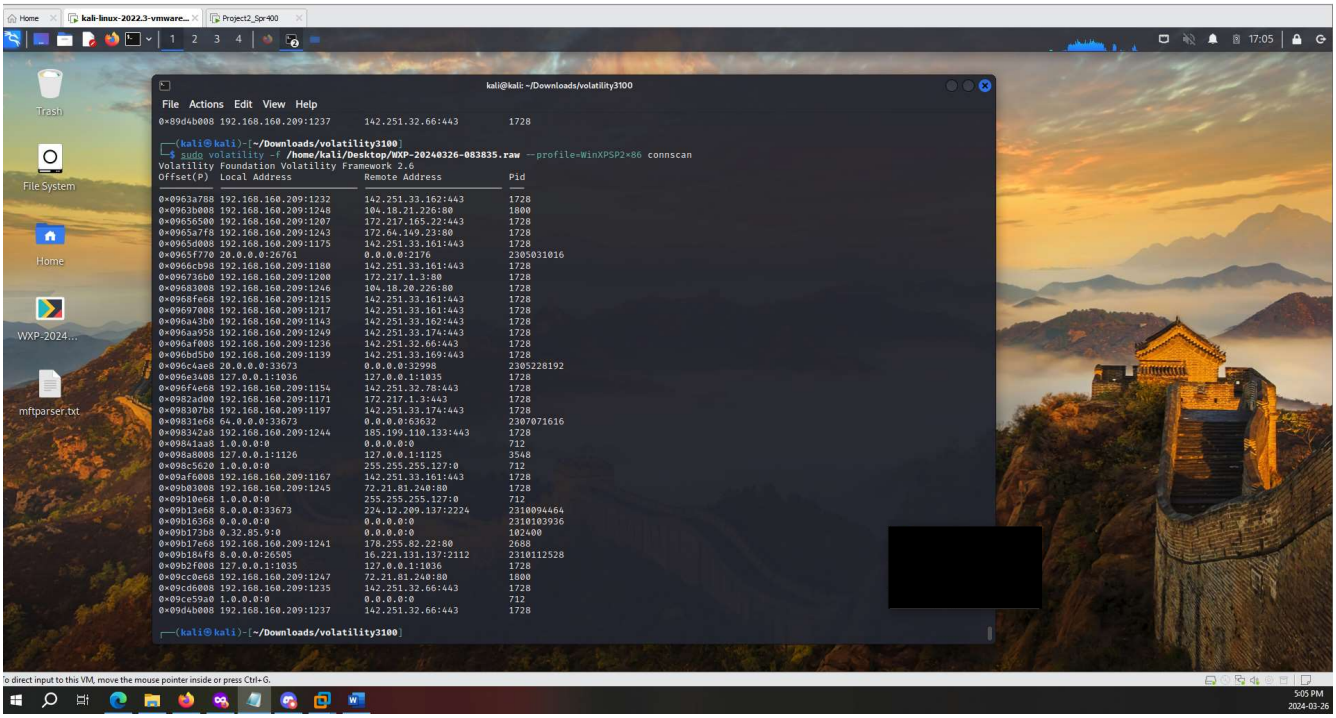


Networking

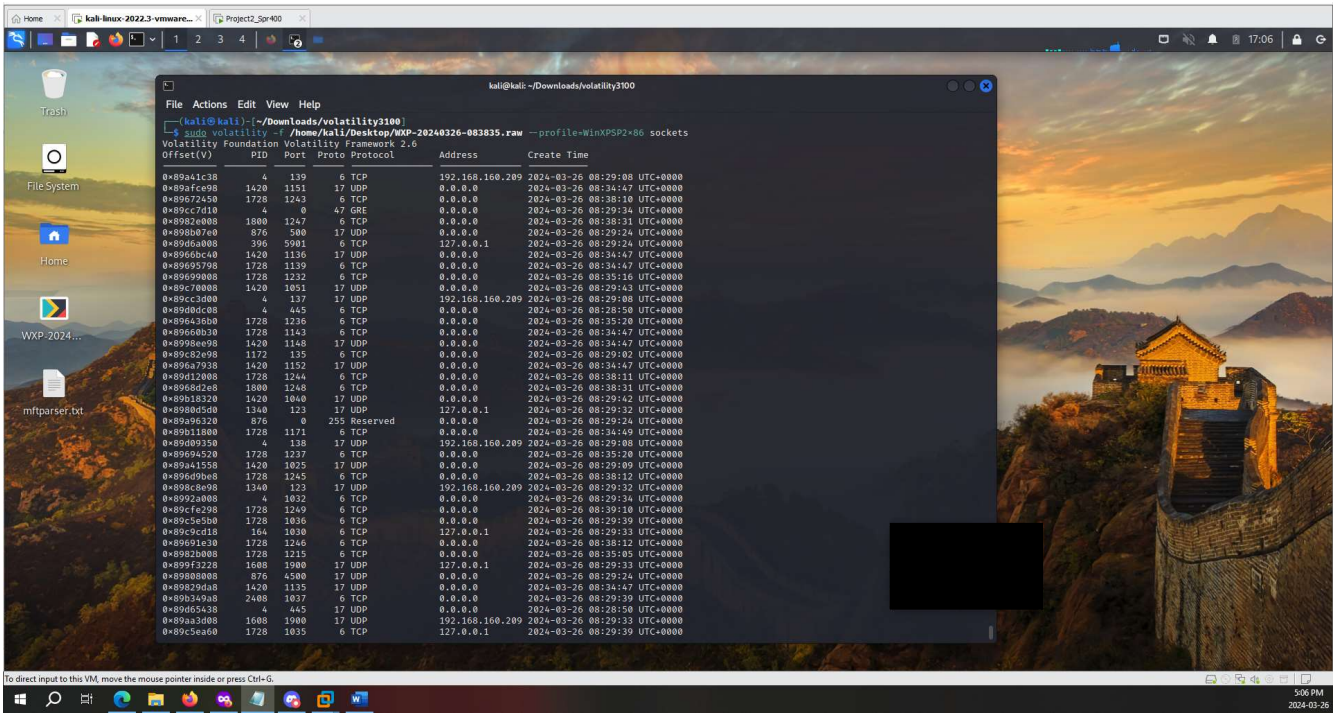
- **connections:** This command views TCP connections that were active at the time of the memory acquisition. For example, you can see various network connections made by different processes. For example, the process with PID 1728 has made several connections to different IP addresses on various ports.



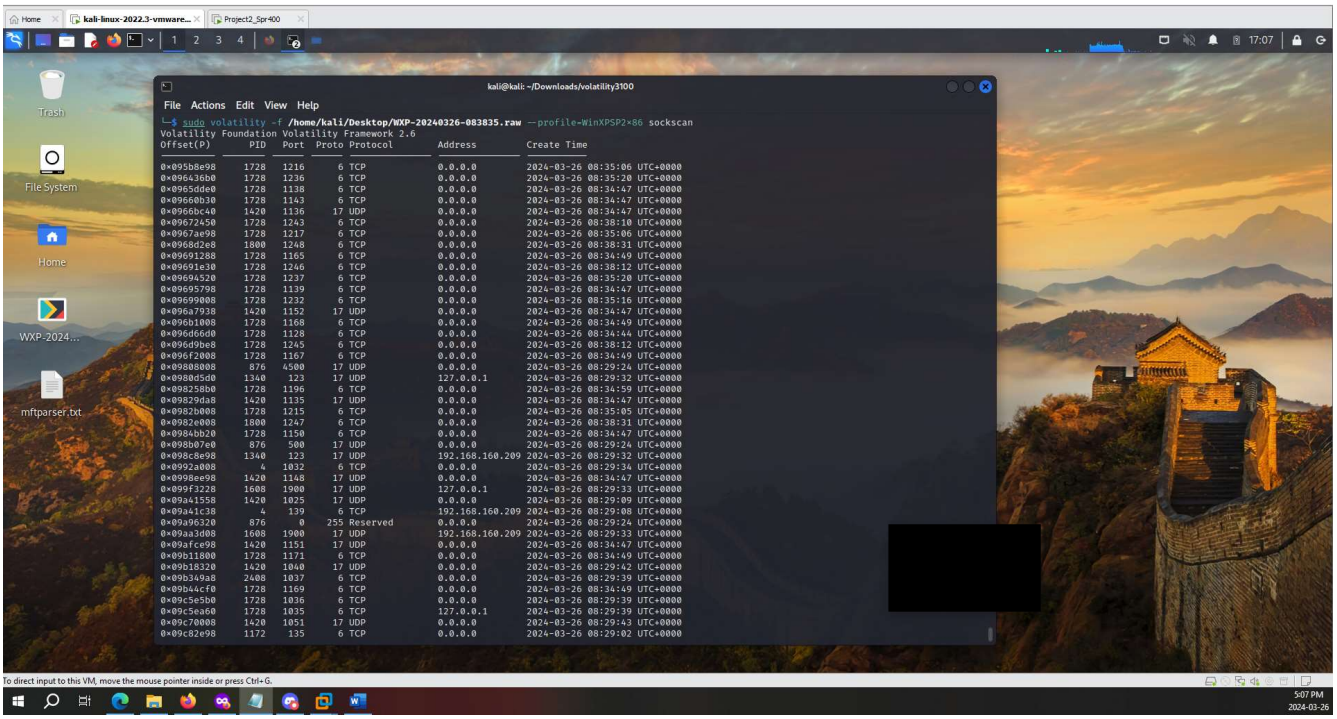
- **connscan:** This command is similar to the previous one, but it can find artifacts from previous connections that have since been terminated, in addition to the active ones. For example, the first row indicates a connection was made by the process with PID 1728 from local address 192.168.160.209:1232 to remote address 142.251.33.162:443.



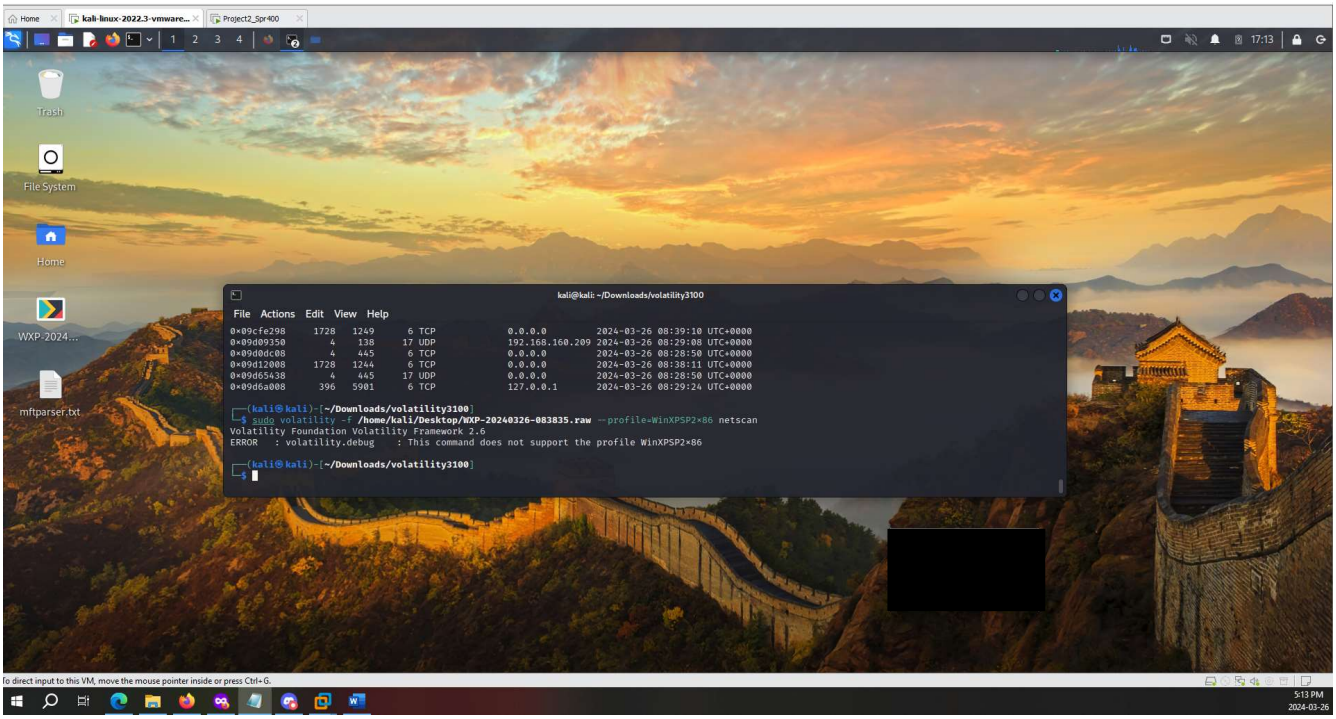
- **sockets:** This command is used to detect listening sockets for any protocol (TCP, UDP, RAW, etc). For example, the first row indicates a TCP socket was created by the process with PID 4 at local address 192.168.160.209 on port 139 at the specified create time.



- **sockscan:** This command is used to find the ADDRESS_OBJECT structures. Each row represents a network socket in the system. For example, the first row indicates a TCP socket was created by the process with PID 1728 at local address 0.0.0.0 on port 1216 at the specified create time.



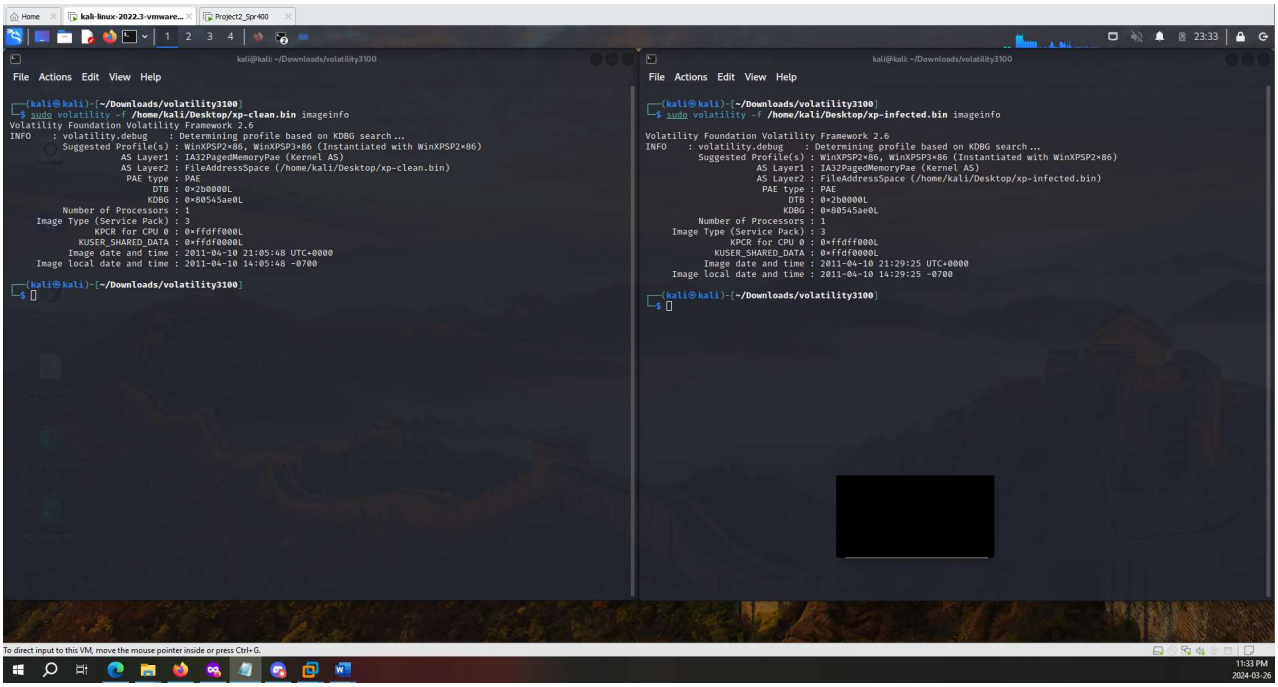
- **netscan:** This command is used to list network-related objects, like sockets and connections. This command did not work because it is designed to work with newer versions of Windows (Vista and later) that include certain networking features unlike my WindowsXP machine. In other words it is not compatible.



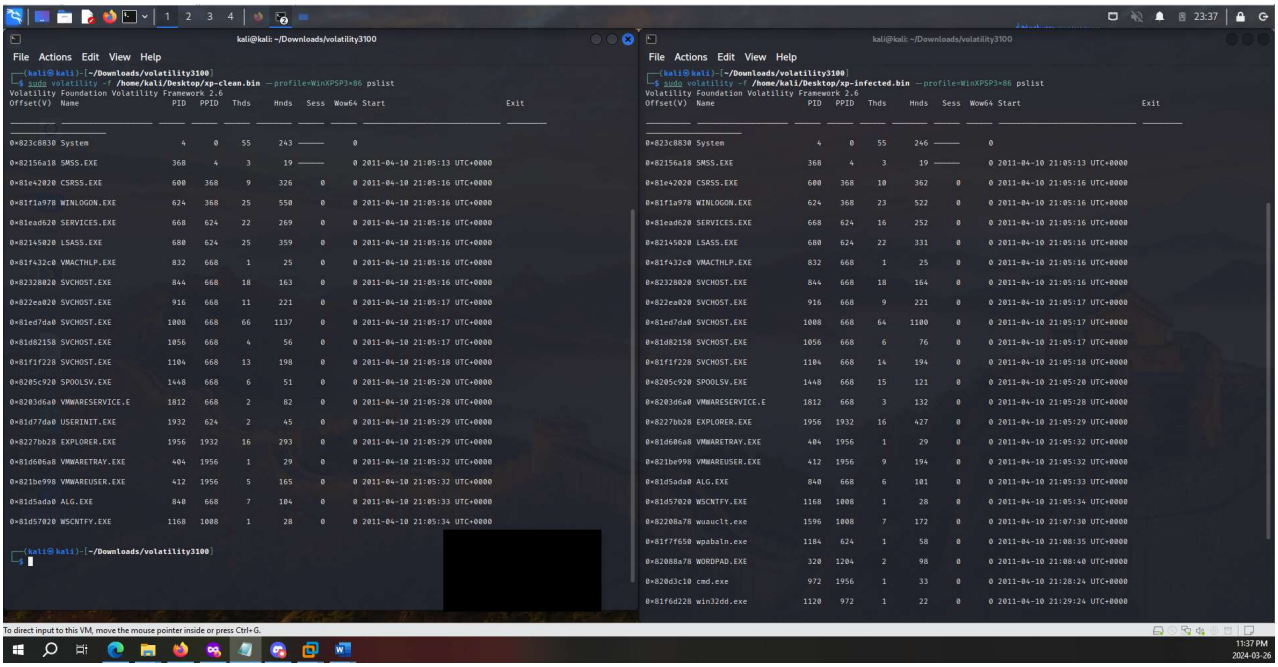
Part 3: Malware Memory Images

This section is similar to the previous one but this time we compare outputs based on the provided clean and infected memory dumps.

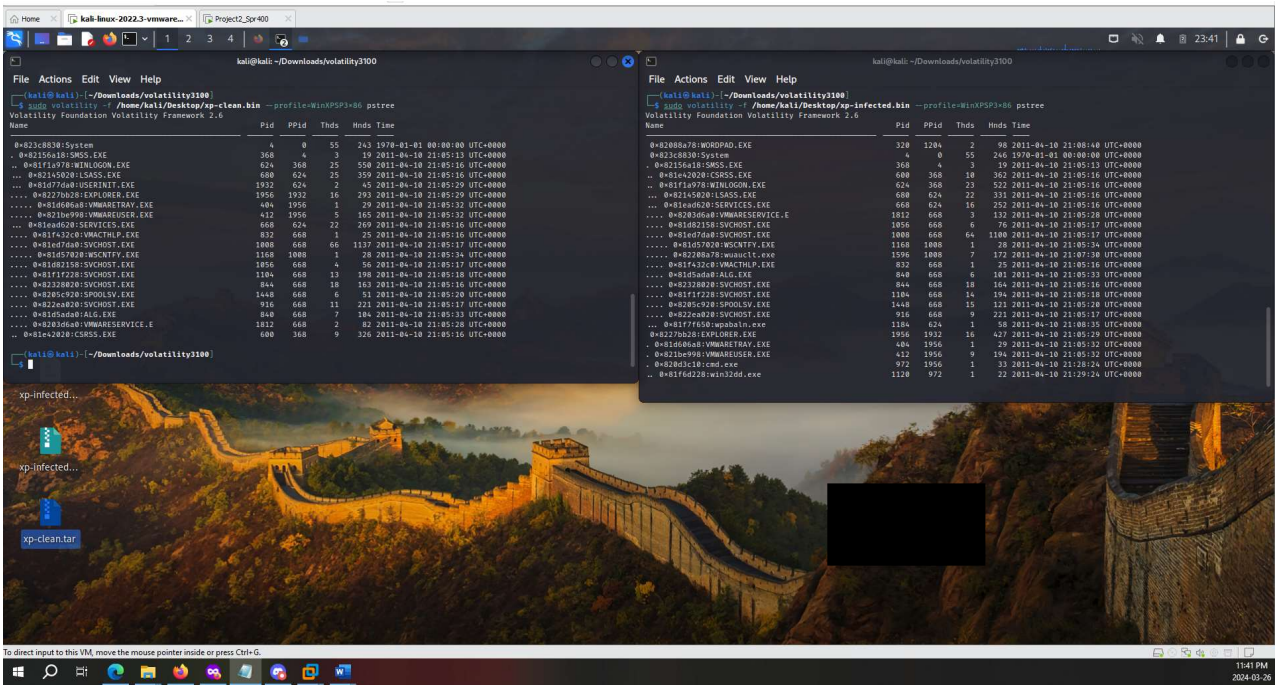
- **imageinfo**: Here we can see that both machines have the same suggested profile by Volatility, indicating that we're dealing with the same operating system in both memory dumps.



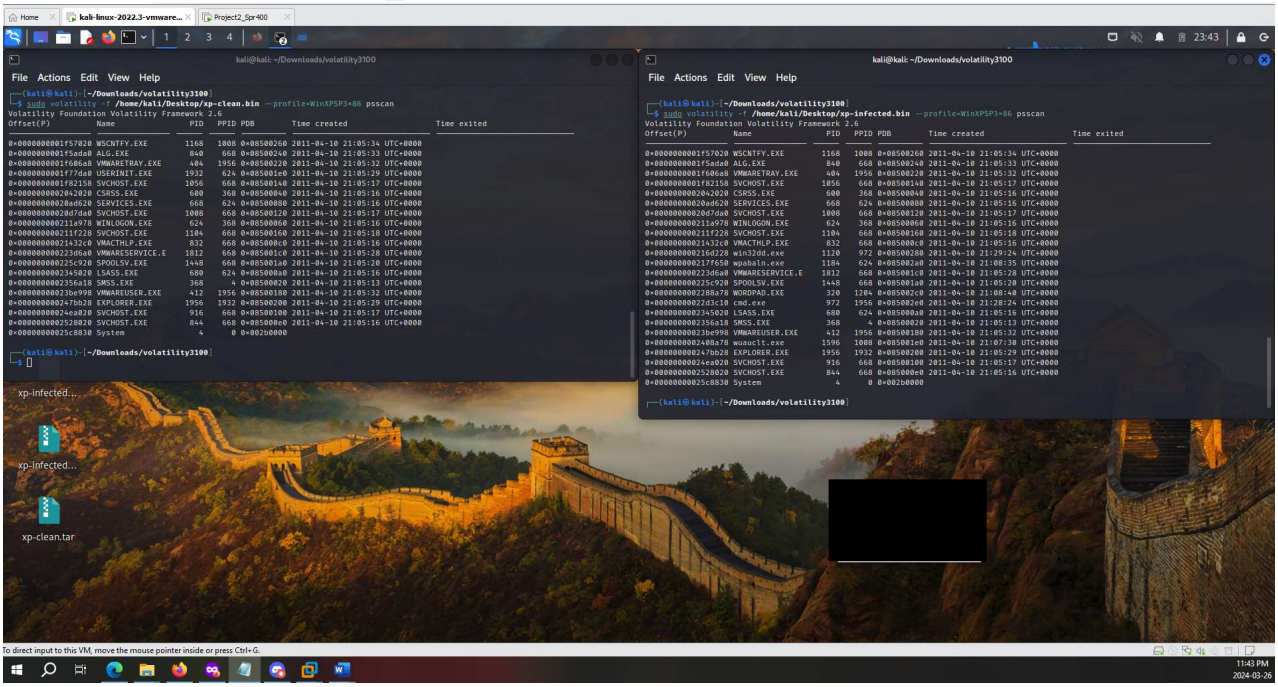
- **Pslist:** Here, you can see me utilizing the pslist plugin to look at the list of processes running on both machines when the memory dump was taken. When comparing both outputs I notice 5 more processes running on the infected machine. These processes are **wuauclt.exe, wpabaln.exe, WORDPAD.EXE, cmd.exe, and win32dd.exe**. We know already that **cmd.exe, and win32dd.exe** are used as part of the process of creating the memory dump, so we can ignore those two. Thus, we're now left with the three other processes to suspect of being malicious: wuauclt.exe, wpabaln.exe, and WORDPAD.EXE.



- **Pstree:** After looking at both outputs for this plugin. I do not see anything standing out to help me narrow down the malicious process.

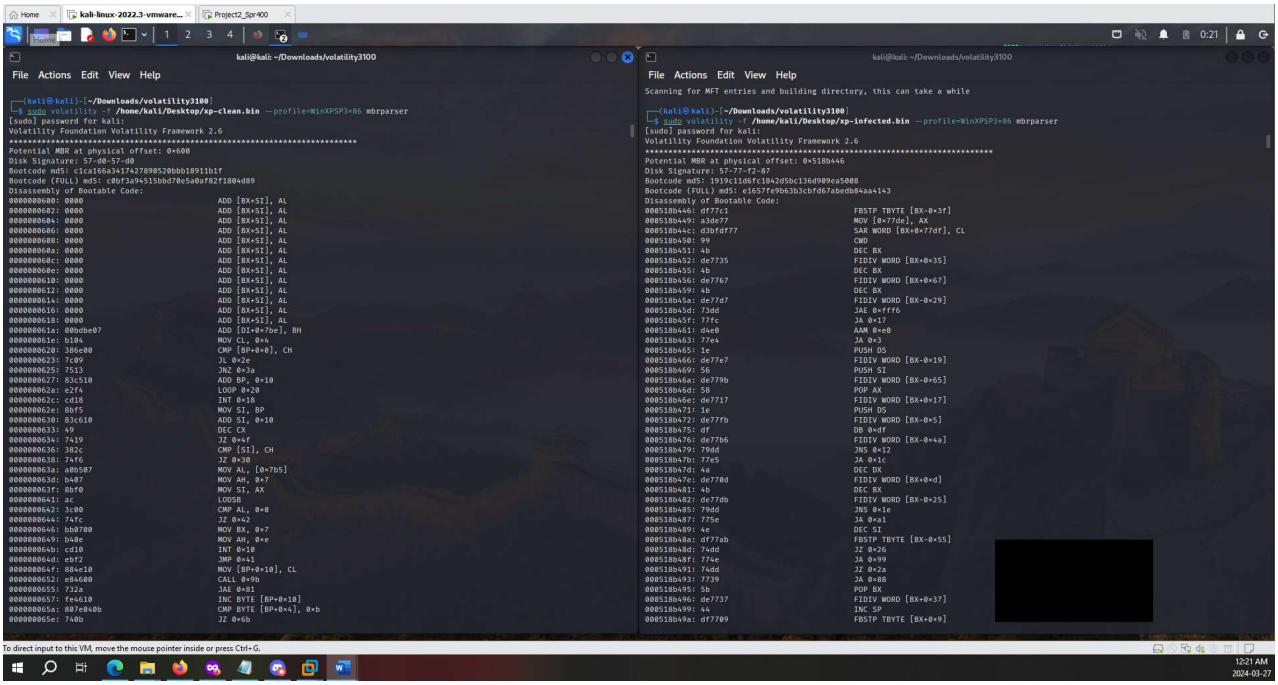


- **Psscan:** Here, you can see that I utilized the psscan plugin with the intention of finding any processes with a “Time Exited” value which would indicate possible hidden processes being created and thus narrowing down our list of suspicious malwares. Unfortunately, there was not any “Time Exited” fields found in the output.

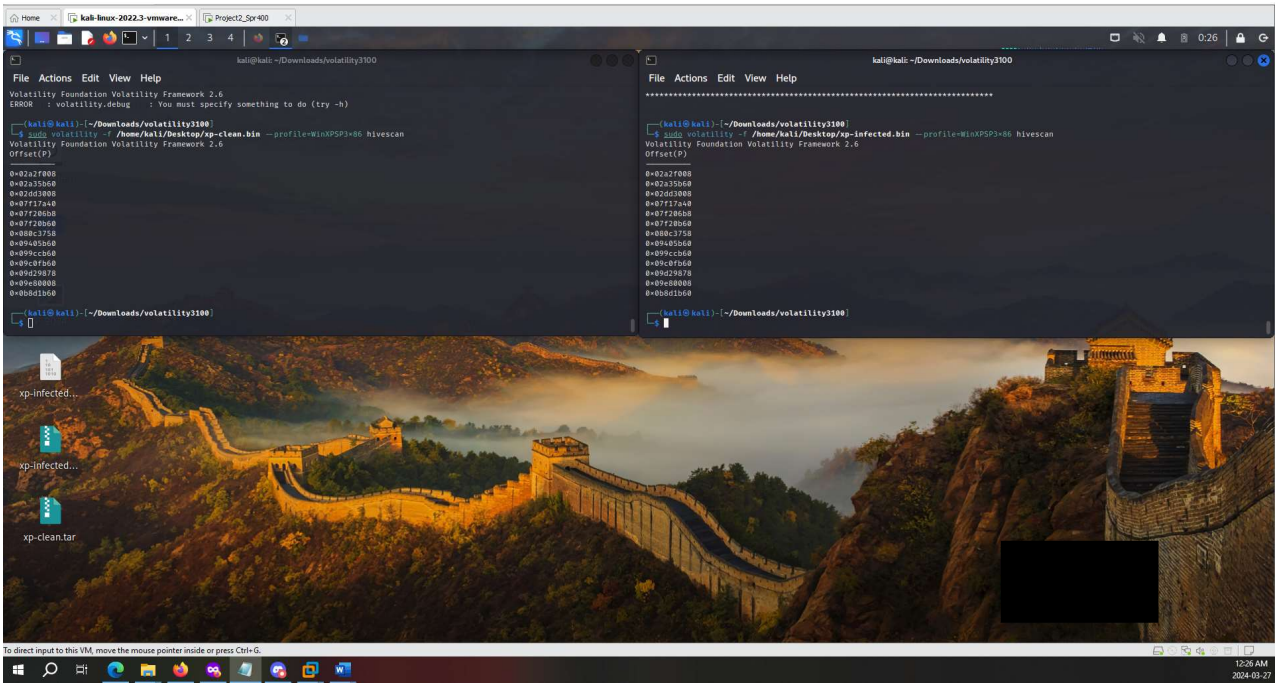


- **Mbrparser:** Here, I initially attempted to run mftparser (as it worked in part2), however this time it did not work in both clean and infected memory dumps, so I used the mbrparser

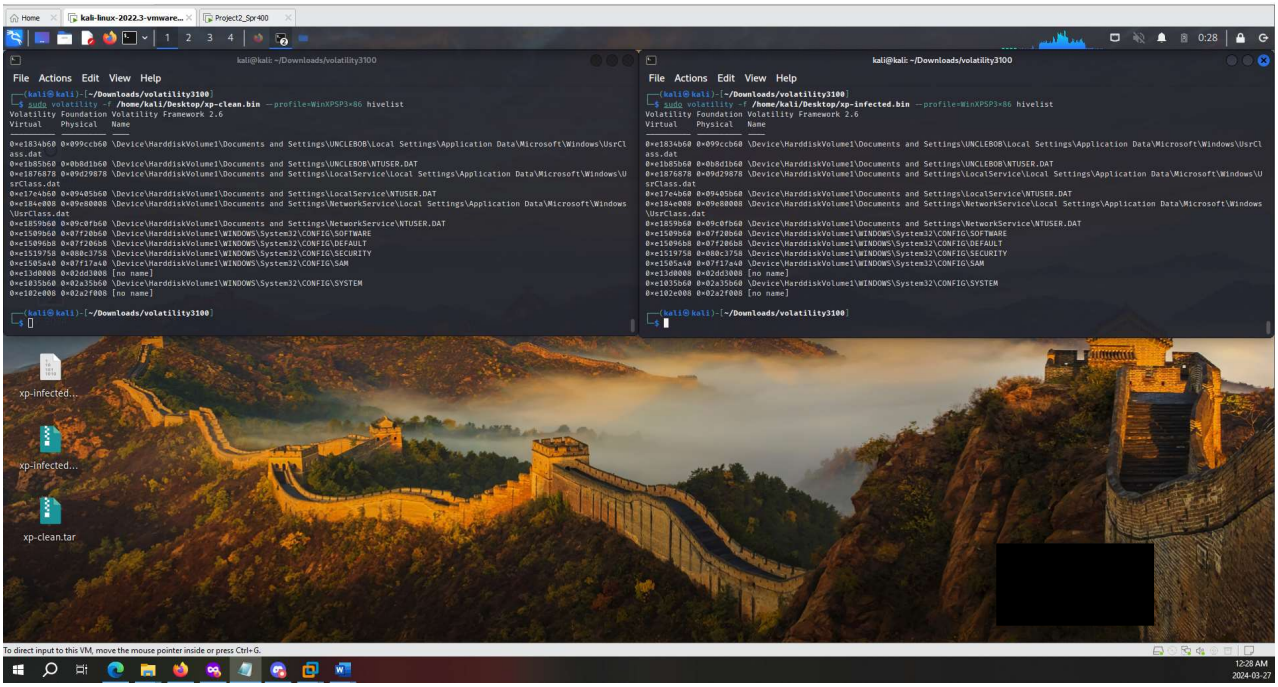
plugin instead. Despite seeing some output with this plug in, there was once again nothing concrete that can be used to better make the decision of isolating a single malware process.



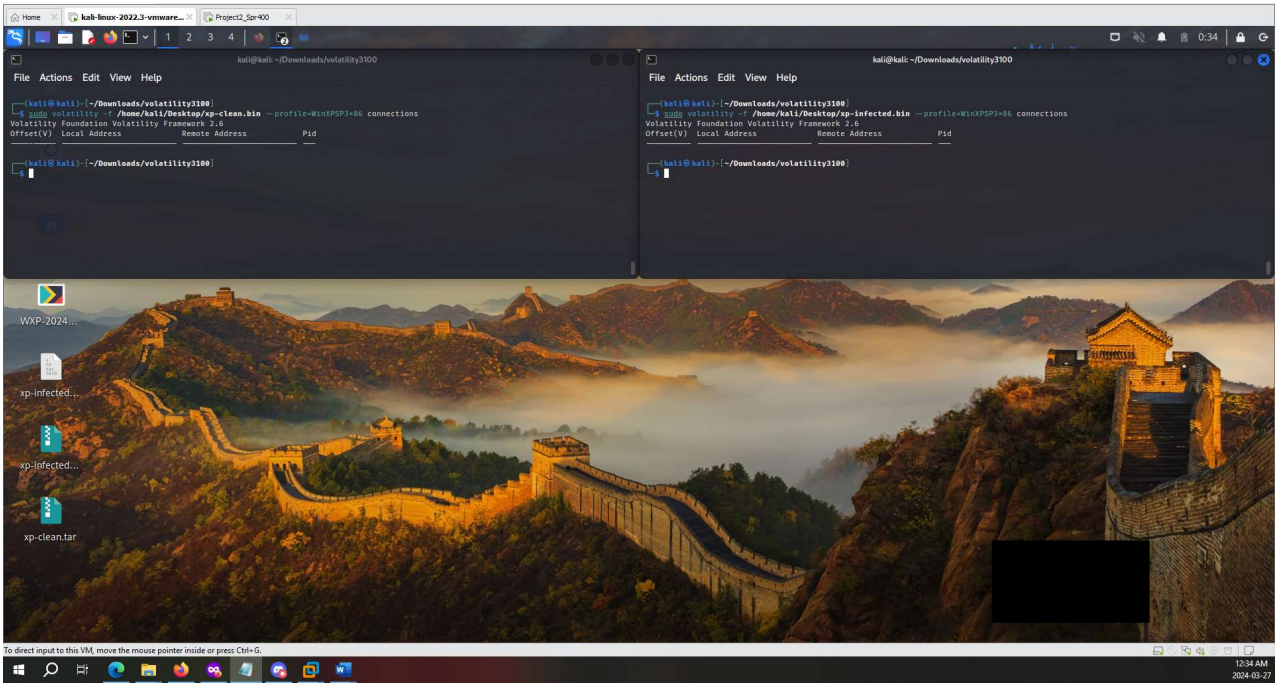
- **Hivescan:** Here, you can see me utilizing the hivescan plugin on both dumps to locate the registry hives in memory. Again, this didn't lead to any concrete conclusions.



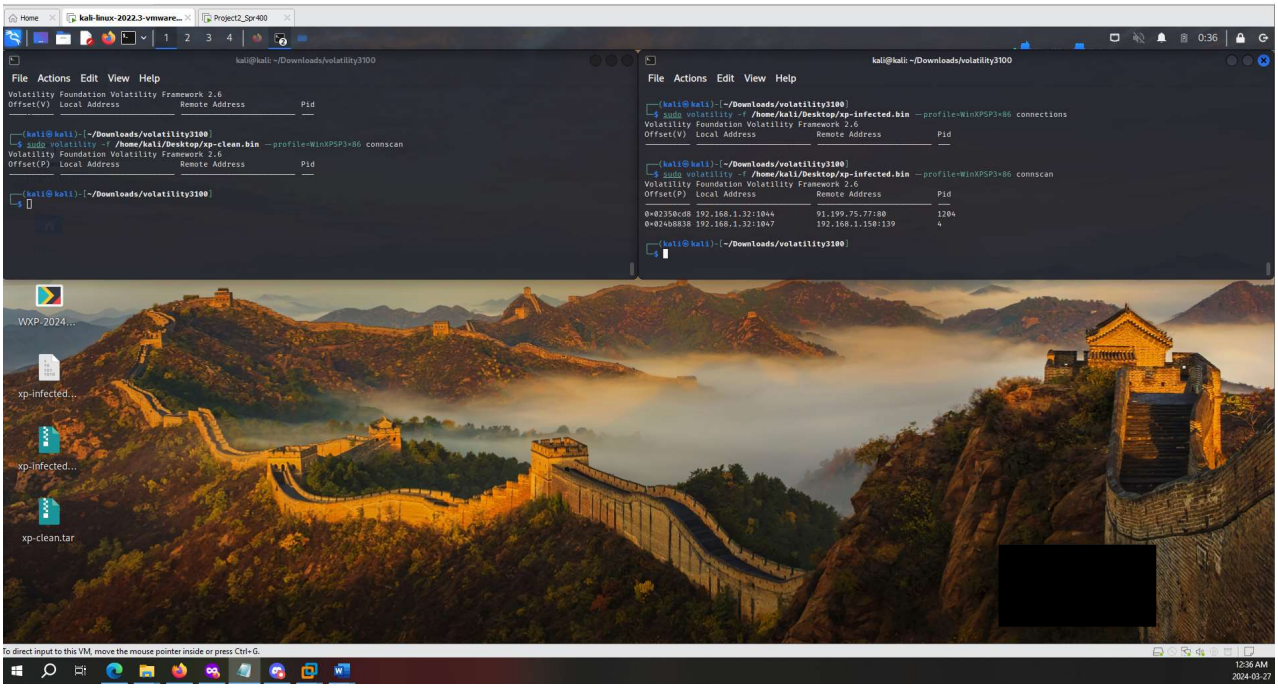
- **Hivelist:** Here, you can see me utilizing the hivelist plugin on both dumps to find the virtual addresses of the registry hives in memory. Again, this didn't lead to any concrete conclusions.



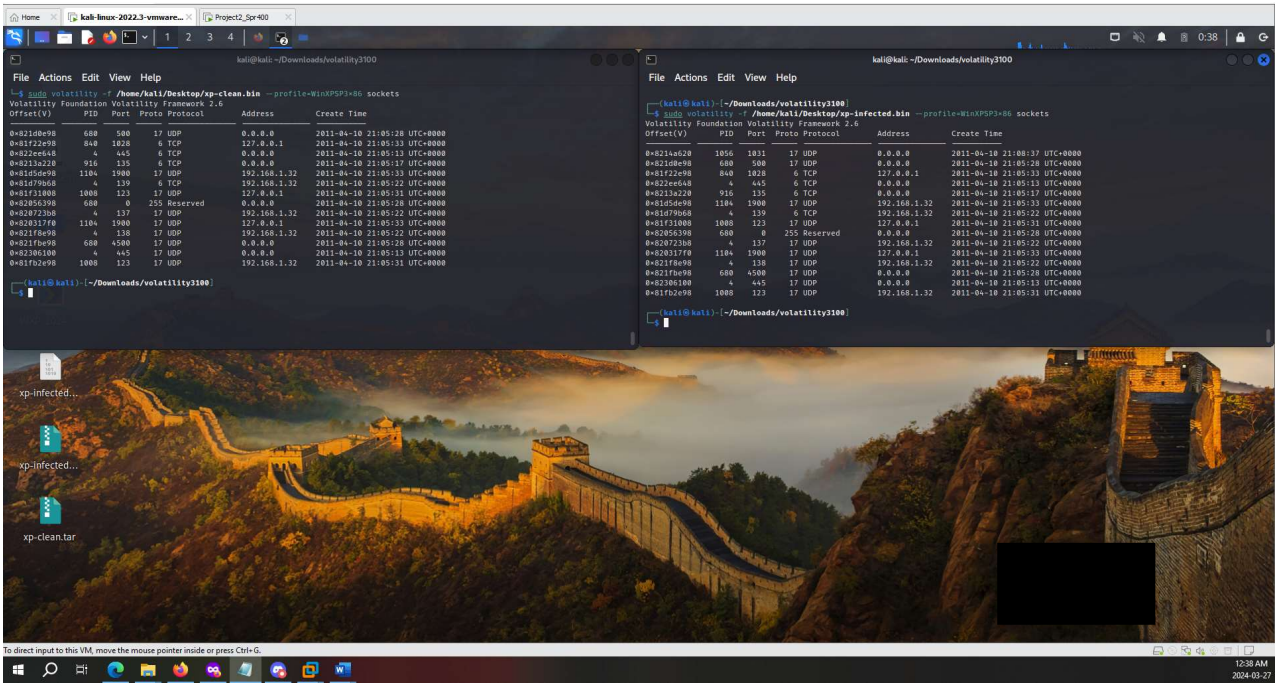
- **Connection:** Here, we can see the output of this plugin on both clean and infected memory dumps showing that there were no active connection at the time these memory dumps were created.



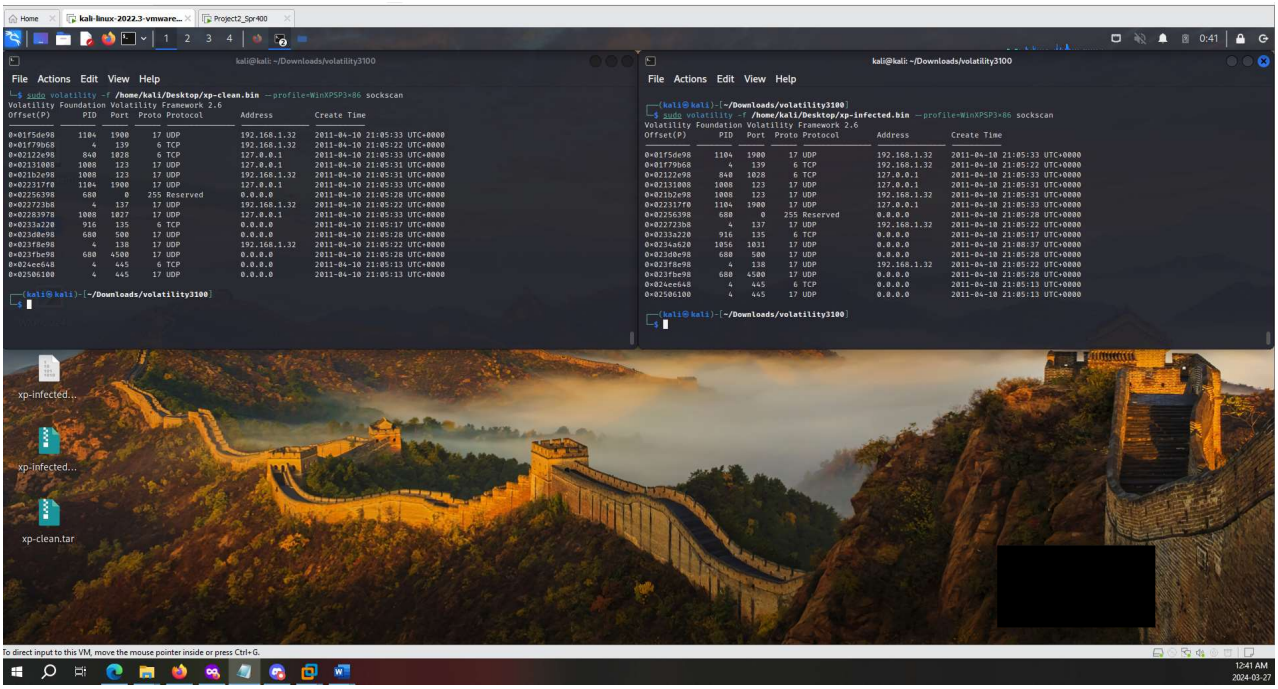
- **Connscan:** Here, we utilized the connscan plugin to determine if there are any connections that we're previously terminated in either machine from which the dump was taken. On the infected machine we realize that we have 2 connections that were created (and later terminated) – one with a process of ID (PID) of 1204 and the other with a PID 4. These processes were created because of that connection. If we take the PIDs and compare them with the list of running processes (pslist) above, we will find that they come back as the Parent process ID (PPID) of two processes – **WORDPAD.EXE** and **SMSS.EXE**, respectively. Recall that WORDPAD.EXE was already one of the processes we were suspecting earlier – with that in mind we can now move forward to utilize the next plugin.



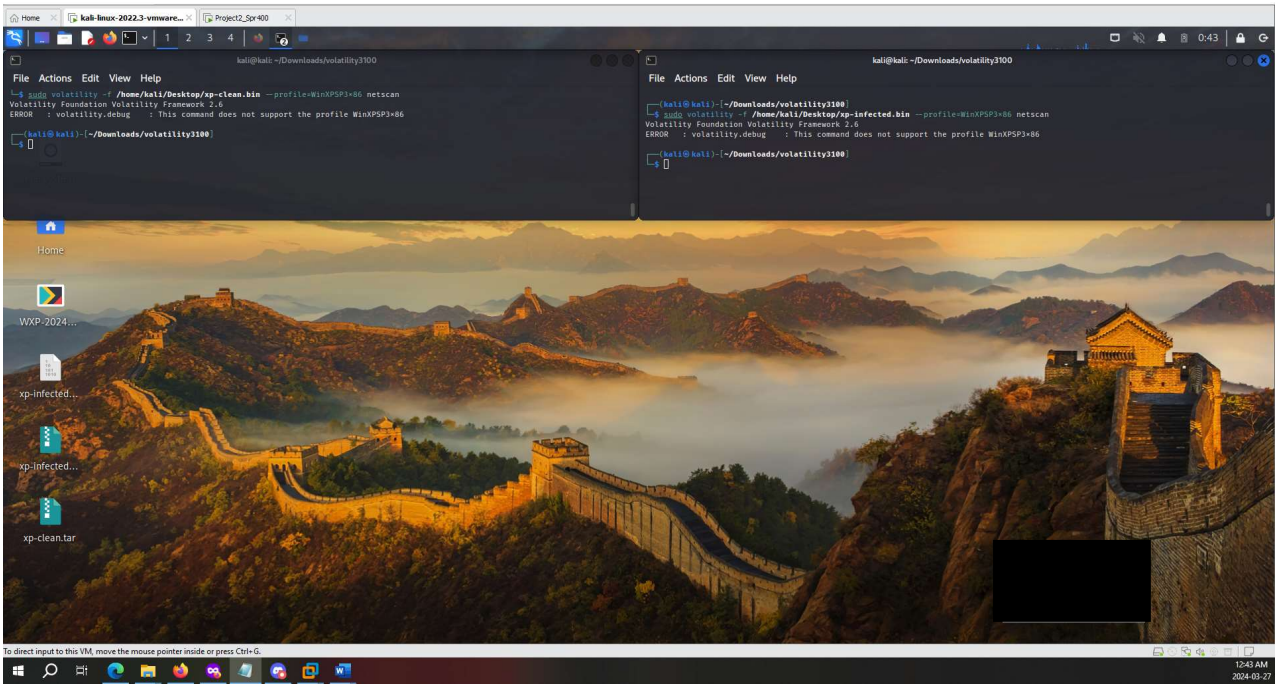
- **Sockets:** Here, I am utilizing the sockets plugin on both clean and infected memory dumps to determine the live sockets that existed at the time of the memory dump. When comparing both outputs they seem almost identical. The only difference is the infected one has an additional process that traces back to svchost.exe. In other words, there are no new leads here we could use to narrow down the list of malware processes.



- **Sockscan:** Here, I am utilizing the sockscan plugin on both clean and infected memory dumps to determine the previously existing sockets. Again, there are no new leads here we could use to narrow down the list of malware processes.



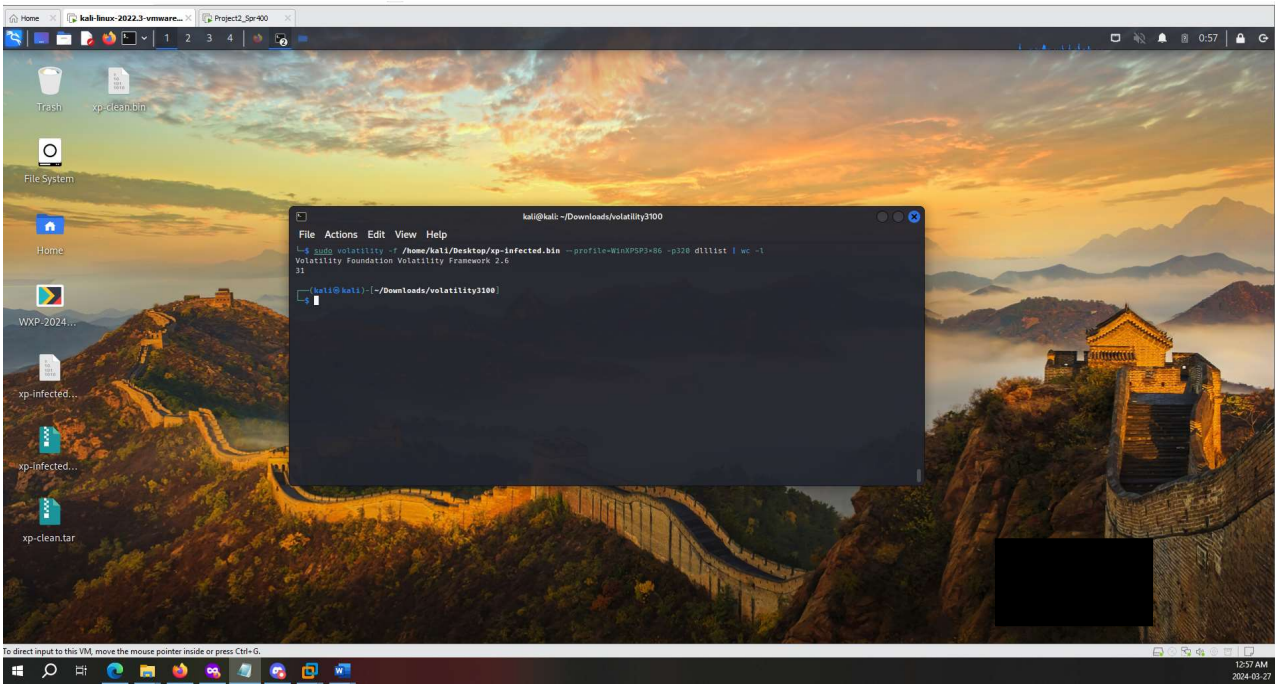
- **Netscan**: like previously explained in part 2, this plugin is not compatible with Windows XP, so we got an error.



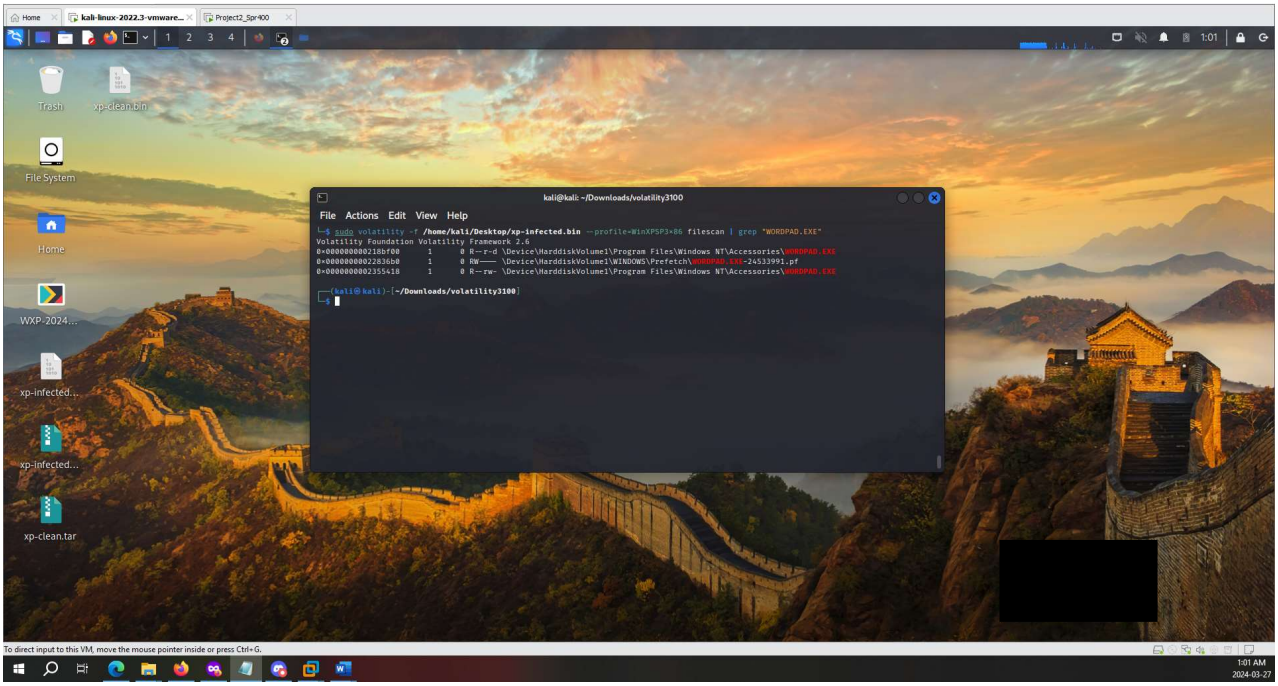
Task 2

Thus far we have suspected a few malicious processes (**wuauclt.exe, wpabaln.exe, WORDPAD.EXE,**) but only one of them was created from a previous connection (WORDPAD.EXE). As per the instructions, in this task we will dig in deeper to find more proof on what makes this process seem malicious. To do this I'm thinking of checking the characteristics of this process against those of the legitimate program to determine if there is anything interesting or worth noting.

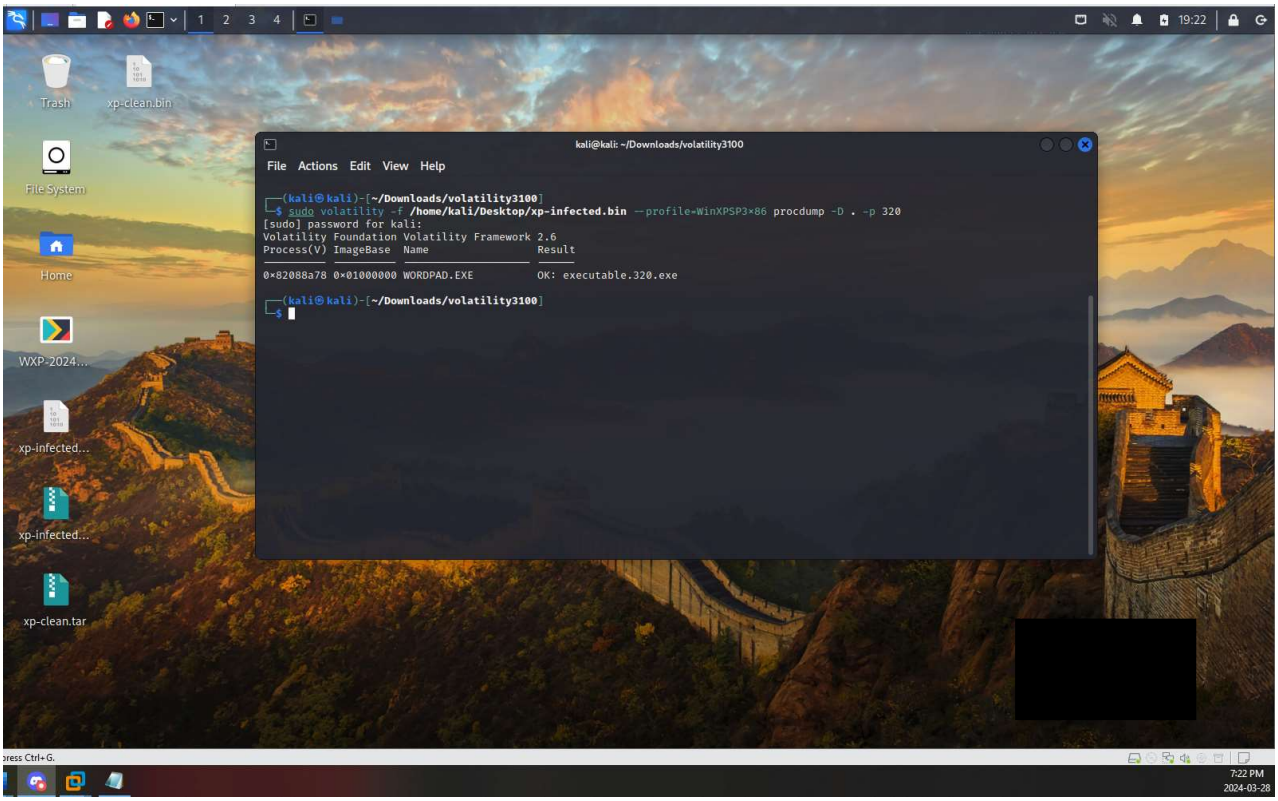
A common practice in digital forensics and incident response is to examine the Dynamic Link Libraries (DLL) of the files associated with a suspect process. We can do this using volatility. Knowing the Process ID of WORDPAD.EXE (320) we can examine the DLLs loaded by this specific process. After running that we see that **31 DLLs** were loaded by this specific process. This is concerning because after doing some research I found that on average, WordPad loads **a few hundred DLLs** during its execution. This finding suggests that WORDPAD.EXE is a suspicious process.



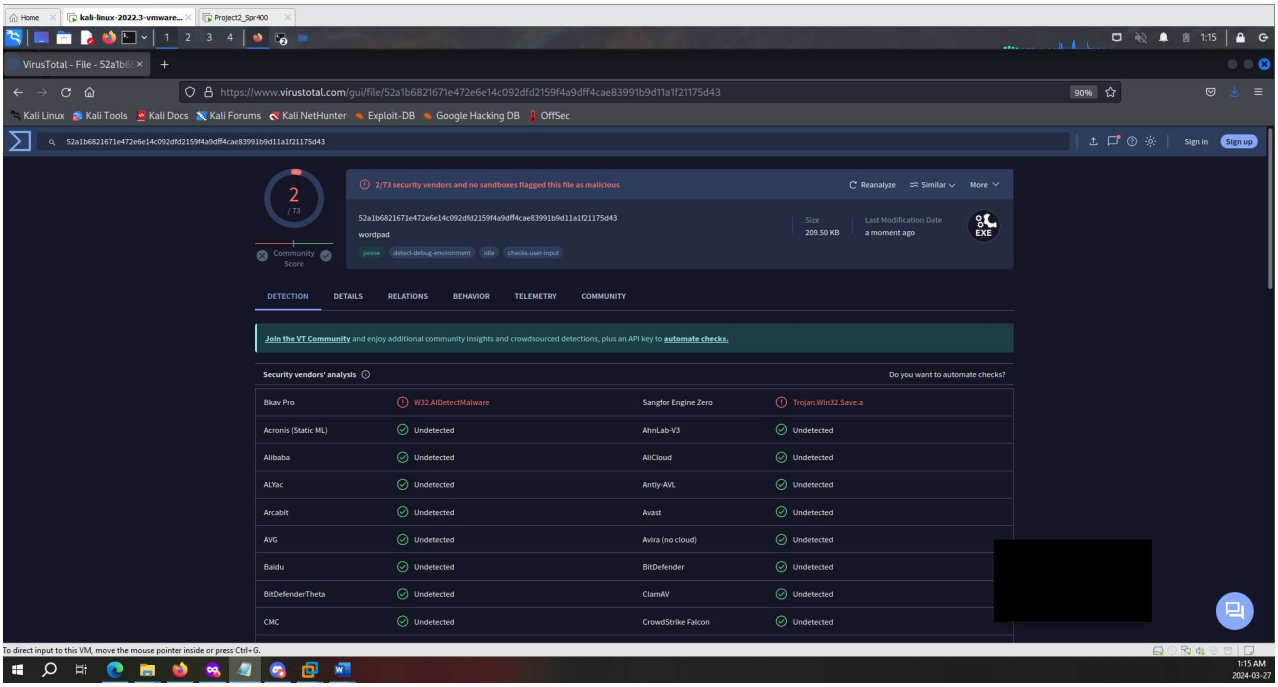
Another thing we can do is utilize the filescan plugin to determine if there were any open files that contain the contain the term WORDPAD.EXE at the time the memory dump was taken. The output shows that we have 3 files open, 2 of which are located in the same directory (\Windows NT\Accessories). This is concerning because there is no reason why there should be two files with the same name (but with different permission) open in the same directory. The third file, (located in \WINDOWS\Prefetch) is not suspicious as it is known to be used to be typically used by the operating system to boost the loading speed of running programs. This is another finding that further confirms our suspicions that WORDPAD.EXE is a malicious process.



Now that our suspicions are leading towards a single process (wordpad.exe), we can utilize the procdump plugin to export this process in an executable format. We can later utilize this executable to upload to an online service like VirusTotal to check it against 70 antivirus scanners and services.



After uploading the executable, we can see in the below screenshot that it was indeed recognized as malicious by two security vendors (“Bkav Pro” and “Sangfor Engine Zero”); which finally concludes our investigation with WORDPAD.EXE being the malicious process.


VirusTotal - File - 52a1b6
https://www.virustotal.com/gui/file/52a1b6821671e472e6e14c092dfd2159f4a9dff4cae83991b9d11a1f21175d43
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking DB OffSec
Sign in
Sign up
2
/ 73
Community Score
2/73 security vendors and no sandboxes flagged this file as malicious
Reanalyze
Similar
More
52a1b6821671e472e6e14c092dfd2159f4a9dff4cae83991b9d11a1f21175d43
wordpad
Size
209.50 KB
Last Modification Date
a moment ago
EXE
DETECTION
DETAILS
RELATIONS
BEHAVIOR
TELEMETRY
COMMUNITY
Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.
Security vendors' analysis
Do you want to automate checks?
Bkav Pro
W32.AIDetectMalware
Sangfor Engine Zero
Trojan.Win32.Save.a
Acronis (Static ML)
Undetected
AhnLab-V3
Undetected
Alibaba
Undetected
AliCloud
Undetected
ALYac
Undetected
Antiy-AVL
Undetected
Arcabit
Undetected
Avast
Undetected
AVG
Undetected
Avira (no cloud)
Undetected
Baidu
Undetected
BitDefender
Undetected
BitDefenderTheta
Undetected
ClamAV
Undetected
CMC
Undetected
CrowdStrike Falcon
Undetected
To direct input to this VM, move the mouse pointer inside or press Ctrl+G.
1:15 AM
2024-03-27