## Forensics tools

Volatility (2.6.1):  
“A single, cohesive framework analyzes RAM dumps from 32- and 64-bit Windows, Linux, Mac, and Android systems. Volatility's modular design allows it to easily support new operating systems and architectures as they are released.”  
<https://www.volatilityfoundation.org/releases>

Volatility-autoruns plugin (no version numbers):  
“Autoruns basically automates most of the tasks you would need to run when trying to find out where malware is persisting from. Once all the autostart locations are found, they are matched with running processes in memory.”  
<https://github.com/tomchop/volatility-autoruns>

FTK Imager (4.5.0.3):  
“FTK® Imager is a data preview and imaging tool that lets you quickly assess electronic evidence to determine if further analysis with a forensic tool is warranted.”  
<https://accessdata.com/product-download/ftk-imager-version-4-5>

Sha1sum (8.30):  
“sha1sum is a computer program that calculates and verifies SHA-1 hashes. It is commonly used to verify the integrity of files. It is installed by default in most Linux distributions.”  
<https://man7.org/linux/man-pages/man1/sha1sum.1.html>

Virustotal.com (March 27th, 2021):  
“Analyze suspicious files and URLs to detect types of malware, automatically share them with the security community.”  
<https://www.virustotal.com/gui>

Renenyffenegger.ch (March 27th, 2021):  
“A site that details specifics about various registry keys, values, and data entries.”  
<https://renenyffenegger.ch/notes/Windows/registry/>

Malpedia (March 27th, 2021):  
“The primary goal of Malpedia is to provide a resource for rapid identification and actionable context when investigating malware. Openness to curated contributions shall ensure an accountable level of quality in order to foster meaningful and reproducible research.”  
<https://malpedia.caad.fkie.fraunhofer.de/>

AbuseIPDB (March 27th, 2021):  
“AbuseIPDB is a project dedicated to helping combat the spread of hackers, spammers, and abusive activity on the internet.”  
<https://www.abuseipdb.com/>

## Create Ubuntu Server VM and install Volatility 2

|  |  |
| --- | --- |
| Ubuntu Server | 20.04.2 |
| CPU | 2 with 1 Core |
| RAM | 4 GB |
| Hard Drive | 30 GB |

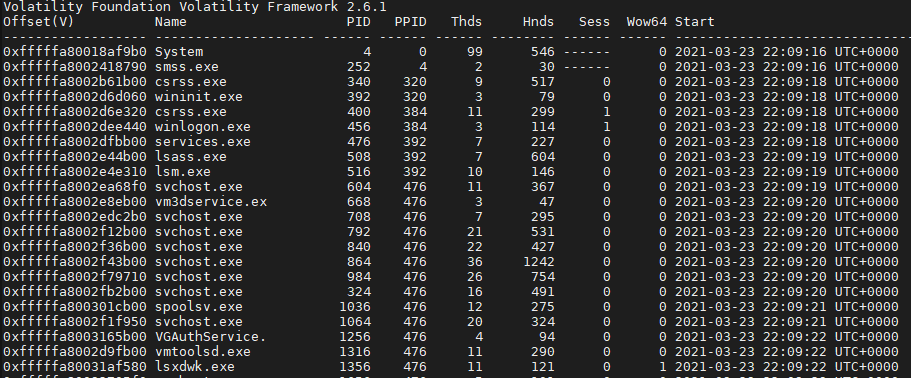
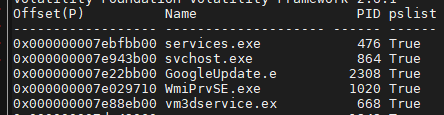
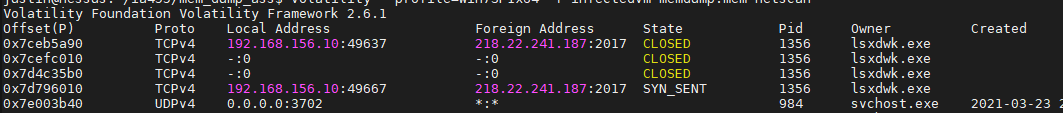
I installed an Ubuntu Server VM with the above specs using otherwise default options. From this point updated the server by running the *sudo apt update -y && sudo apt full-upgrade -y* command. Next, I Volatility-Phocean (2.6.1) via Snap by running the *sudo snap install volatility-phocean -y* command.

## Capture memory with FTK Imager

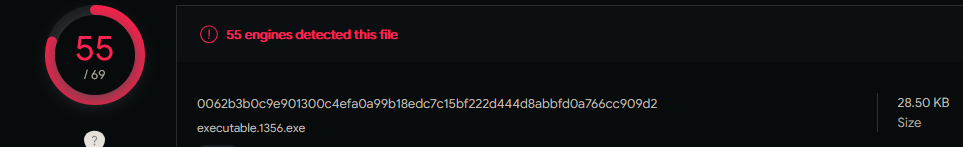
I installed FTK Imager on a USB drive using the following steps:

1. Installed FTK Imager on my host machine.
2. Inserted an NTFS formatted flash drive.
3. Copied the entire "**FTK Imager**" installation folder located at "**C:\Program Files\AccessData\FTK Imager**" to my flash drive.
4. Copied msvcp120.dll, vcruntime140.dll, and all MFC\* files located in **C:\Windows\System32**
5. Attached the flash drive to the Win7-VICTIM VM.
6. Run FTK Imager.exe (as Administrator) and used: File > Capture Memory
7. I made sure that the Include pagefile option was selected and entered a writeable network drive location in the Destination path. I named the capture infectedvm-memdump.mem
8. I verified the sha1 sum of both files before and after transfer to the Ubuntu server.

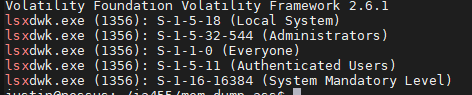
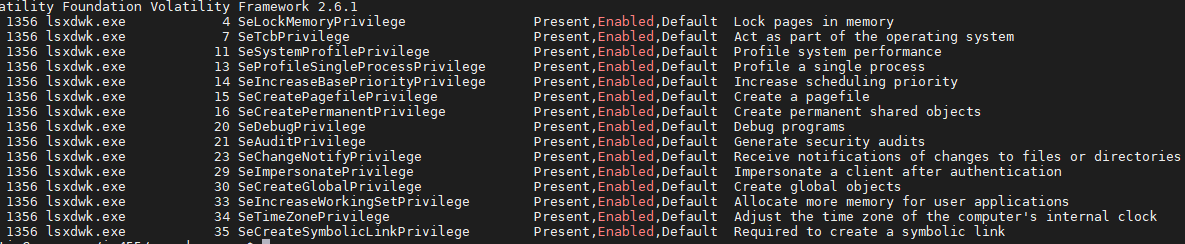
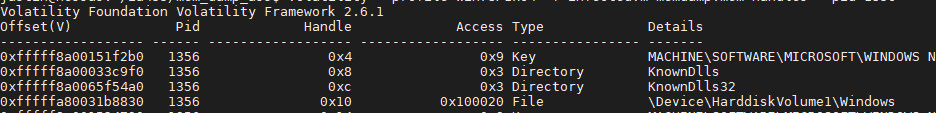
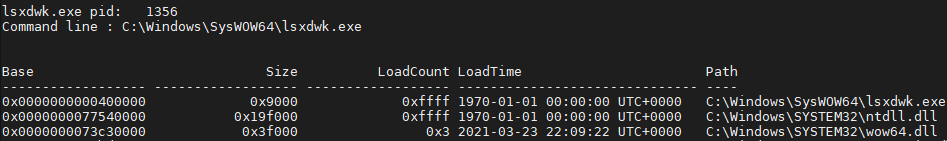
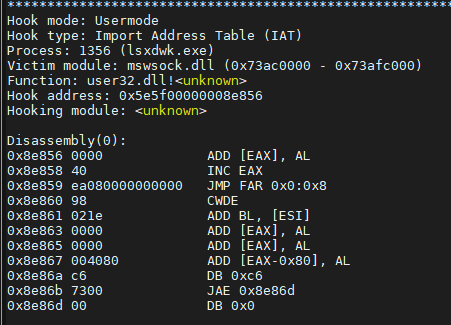
## Analyze memory dump

The first goal that I needed to accomplish was to figure out the correct profile. Since I wasn’t sure of the update status of WIN7-VICTIM, I ran the *volatility* *-f infectedvm-memdump.mem imageinfo* command which determined that the best profile for me to use was **Win7SP1x64** profile (capitalization is important). All listed commands from here on will have been prepended with *volatility --profile=Win7SP1x64 -f infectedvm-memdump.mem* and appended with *| tee example\_name.txt* which sends a copy of the displayed information to a file called example\_name.txt for later. For instance, the command *volatility --profile=Win7SP1x64 -f infectedvm-memdump.mem pslist | tee pslist.txt* will be written simply as *pslist* for the sake of brevity in this report. Additionally, the full file will be included as an addendum to this report. My next step was to get an idea of the system state, specifically, I wanted to know the active processes. I accomplished this with the *pslist* command which gave me a list of active processes:  
  
  
  
This provides me with a lot of information already: virtual memory offset (location of the process in vmem), executable name, PID, Handles, Start & Exit times, etc. Some malware will obscure itself by attempting to decouple its processes from the standard process listing options, so I ran the *psxview* command, which showed that there were no unlisted processes:  
  
  
My next course of action was to look into active network connections, which in a Windows 7 memory dump is accomplished with the *netscan* command. This showed that a process, lsxdwk.exe (PID: 1356) was making some suspicious connections to a foreign address:  
  
  
  
This is suspicious for a couple of reasons: TCP port 2017 which per iana.org is registered to cypress-stat, which is associated with a wireless pneumatic thermometer. This isn’t something that the workstation should be connecting to. Additionally, we can see that the process is actively obscuring its creation time and date. Checking for 218.22.241.187 on abuseipdb.com shows this address is associated with a Chinese ISP (chinatelecom.com.cn) located in Suzhou, Anhui, People’s Republic of China. While there are no reports of recent abuse on this IP address, there’s also no reason for this computer to be connecting to a Chinese IP address. Additionally, the APT 26 (Turbine PANDA, Hippo Team, etc) is known to have been operating out of that area as recently as 2019 and has been associated with this malware (<https://malpedia.caad.fkie.fraunhofer.de/actor/apt_26>), so this could even be a remnant of their operations. With this information I had enough information to correlate processes to PIDs, memory offsets, and handles, so my next course of action was to run the malfind command which will search for suspicious software forms (hidden, injected, obscured code, etc.) and dump any suspicious sections. I wanted to check lsxdwk.exe so I ran the *malfind -D malfind/* command (the -D /folder/location/ command specifics a place to Volatility to export process dumps). This unfortunately didn’t turn up any particularly interesting information.

## Extract and analyze malware

My next step was to carve out the existing process for testing, which I accomplished by running the *procdump --pid 1356 --dump-dir temp/* command which gave me executable.1356.exe (SHA1: c6809c1301c7bd3985e2f915092f235965009500):  
  
   
  
Checking this file on virustotal.com gave me a lot of results, for instance:  
  
  
  
  
So this appears to be a Remote Access Trojan that would grant the attacker a backdoor onto the system which would be in the Initial Compromise, Execution, or Persistence stages of the MITRE ATT&CK framework. Given that this memory image was taken immediately after a reboot, there must be some form of persistence, suggesting it’s at or past the persistence stage of the attack.

## Further information gathering

Now I wanted to see where the file was located on the system, so I ran the *filescan­* command which showed that it was located in **C:\Windows\SysWOW64\lsxdwk.exe** and was about 29kb in size. More information about this malware is needed, so I proceeded to collect related Security Identifies (SIDs) by running the *getsids* command:  
  
  
  
This suggests that the malware has escalated privileges into the Administrators and even possibly the System level. To look into this further, I ran the *privs --pid=1356 | grep Enabled* command to get the privilege tokens assigned to pid 1356:  
  
  
  
This shows that the file can act as part of the OS, create objects, track file system changes, impersonate clients, alter memory allocations/status, debug programs, and more. From here, the next course of action was to determine for far the file had spread itself in the system, I ran several commands:  
*handles --pid=1356* which shows the files, keys, threads, and processes that lsxdwk.exe has opened:  
  
  
 *dlllist --pid 1356* which shows which DLLs are being accessed by lsxdwk.exe:  
  
   
  
With this information, it was possible to dump the API hooks that PID 1356 was associated with via the *apihooks --pid 1356* command. There were many such entries, but an interesting example is:  
  
  
  
Here we can see that lsxdwk.exe has victimized mswsock.dll (which is used for managing sockets). Next, I wanted to see if lsxdwk.exe had any special environment variables set up, so I dumped this information using the *envars --pid 1356* command to show only the specified process’s environmental variables. I then dumped the environmental variables of lsxdwk.exe’s parent process (pid=476) to get a reasonable comparison by running the *envars --pid 476* command and comparing the results. The envars of lsxdwk.exe have several additional entries:

* APPDATA **C:\Windows\system32\config\systemprofile\AppData\Roaming**
* LOCALAPPDATA **C:\Windows\system32\config\systemprofile\AppData\Local**
* USERDOMAIN WORKGROUP

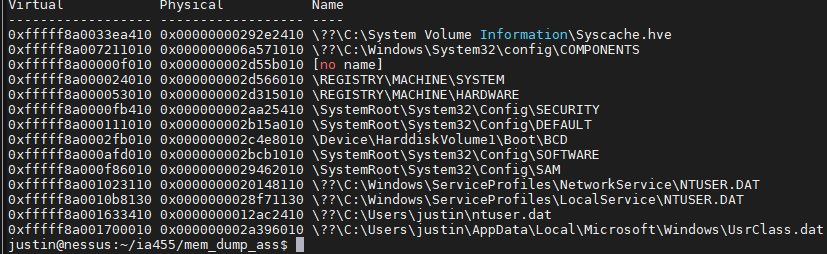
Likely the addition of Roaming and Local in the AppData folder is to give the malware a working directory if needed, many programs use this space for this exact purpose; therefore, this is likely an attempt to mimic normal behavior to disguise malicious actions. More concerning is that it looks like the malware attempts to work in the domain workspace, likely for network reconnaissance and lateral movement. Note that this machine is just part of the default workgroup rather than attached to a domain, so the threat is lower as no other machines are in this workgroup.

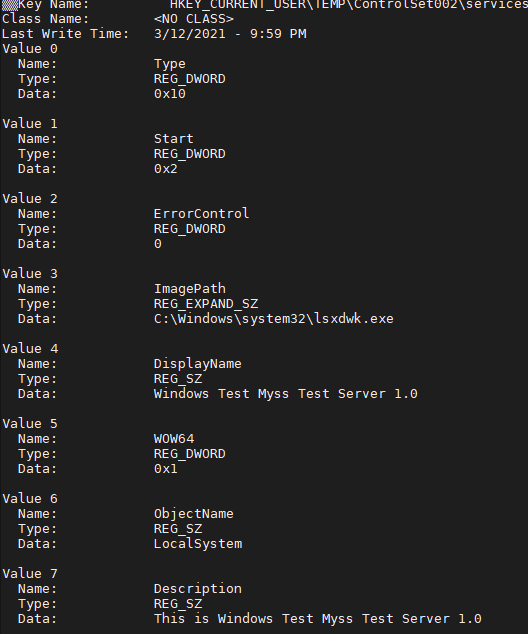
## Finding persistence mechanism:

To figure out how the malware was persisting between reboots, I first wanted to check the autoruns, which entailed installing a volatility plugin called “Autoruns”. The process for installing it was quite simple:

1. Download the plugin from Github with the *wget* [*https://github.com/tomchop/volatility-autoruns/archive/refs/heads/master.zip*](https://github.com/tomchop/volatility-autoruns/archive/refs/heads/master.zip)command.
2. Extract using the *unzip master.zip autoruns/* command
3. Use **--plugins=autoruns/volatility-autoruns-master/** when running commands

With the plugin installed, I ran the *autoruns* command which unfortunately didn’t have any entries for lsxdwk.exe.

Since I couldn’t find anything compelling in autoruns, I decided to check the next most logical source, the registry. It took several commands to get at specific registry keys, values, and data:  
First, I ran the *hivelist* command which shows the top-level hives and their memory offsets:  
  
Next, I ran the *hivedump -o 0xfffff8a000024010* command (note the 0xXXXXXXXXX after the -o flag is the virtual memory offset to the hive to be dumped) to dump the SYSTEM hive as that is a place that services are often enumerated and has a persistence mechanism built into it. I then used the *less 14-lsxdwk\_SYSTEM\_hivedump.txt | grep lsxdwk* command to search for any keys containing the malware’s name, which gave me this key:  
  
To see that hive, I next ran the *dumpregistry -o 0xfffff8a000024010 --dump-dir temp/* command which carves out a .reg file from the selected virtual memory offset location and stores it in the specified --dump-dir directory. Using *cat* lsdwk\_exported\_registry\_SYSTEM\_keys.txt to view the output I saw this:

  
Note that Value 1, named ‘Start’, contains the REG\_DWORD data value of 0x2 which per [renenyffenegger.ch](https://renenyffenegger.ch/notes/Windows/registry/tree/HKEY_LOCAL_MACHINE/System/CurrentControlSet/Services/_driver_name_/index) means: start the service automatically via smss.exe or services.exe. I was able to cross-reference this with the pslist info to find that lsxdwk.exe had a PPID of 476, which matches with services.exe (see the pslist screenshot). So, this is how the malware maintains persistence.

## Timeline

Finally, I wanted to get both an lsxdwk.exe specific timeline as well as a complete timeline for the whole memory dump. This was accomplished first by using the *timeliner* command to generate a full timeline (see timeline.txt for the full document) which showed a timeline of all that happened from start to finish in that memory dump. To cut it down to just the lsxdwk.exe I ran the *grep -e lsxdwk -e 1356 > lsxdwk\_timeline.txt* command to send all lines with ‘lsxdwk’ or ‘1356’ in them to a new document ()see lsxdwk\_timeline.txt for full document). This timeline shows the order in which lsxdwk performed tasks such as registry editing, loading DLLs, and possible process injection via PE DEBUG commands.

## Appendix 1: File list

|  |  |
| --- | --- |
| File Name | SHA1 Hash |
| infectedvm-memdump.mem | b8ffd436831ecd64c7ad3da9f9b3f81a31b3447b |
| pagefile.sys | 425c1bc9cd120d94268a3be5cdd8edc3a3eea43c |
| 01-lsxdwk\_pslist.txt | da8ec258654c61698599f1e038e636e94be646e6 |
| 02-lsxdwk\_psxview.txt | 70bbd340d4308e2ff7ab6434c165916104973a93 |
| 03-lsxdwk\_netscan.txt | 45f1fe43d68bb941465c88efbd76961f278fc1de |
| 04-lsxdwk\_malfind.txt | 4d33e003aad75d7e93ba4907d9af0e202f56a2d5 |
| executable.1356.exe | c6809c1301c7bd3985e2f915092f235965009500 |
| 05-lsxdwk\_filescan.txt | b19db7949917dfa128f365f4adede2d16779def0 |
| 06-lsxdwk\_sids.txt | 785c5f3ed1dc06ee41d9fe23994a4d31e273c064 |
| 07-lsxdwk\_privs.txt | 44ff46f0504540476ca3dbf128659bef41b7c10c |
| 08-lsxdwk\_handles.txt | 2bc3aa7f0f1cc89815821483c4d440ea4689d1a7 |
| 09-lsxdwk\_dlllist.txt | 9d549af7a1cf2c2c6b7b722125155988133251ac |
| 10-lsxdwk\_apihooks.txt | 092a63b01ba227507bbeb92852cf78c4140e7df7 |
| 11-lsxdwk\_envars.txt | 54392a4c0910551437a2aaa93f81c2bed01340b1 |
| 11-services\_envars.txt | 96faea2d6aab2fa3745493b2324efd69b901f79f |
| 12-lsxdwk\_autoruns.txt | f2f2c69c9c2b8d94c7e1b15c4e374cade037c17b |
| 13-lsxdwk\_hivelist.txt | 158dc6cc44a9583119808865981eb72c48f80cb9 |
| 14-lsxdwk\_SYSTEM\_hivedump.txt | 3e69be4f030452fbadb08f05206cae634294975b |
| 15-lsxdwk\_dumpregistry.txt | 1afa27dbb471dbbfe362c18c73ba52525f4ab000 |
| 16-lsxdwk\_HKCU\_TEMP\_ControlSet002\_services\_WindowsTestMyss.txt | ad0649e9df010e0d958f3d36c9a4c275c82dbc14 |
| 17-lsxdwk\_full\_timeliner.txt | 6bda2f37ded6f6bf98ce57a091045d9cbfa862fb |
| 18-lsxdwk\_scoped\_timeliner.txt | d31e505b37cdf4ce9b3d63276430504abe46eb27 |