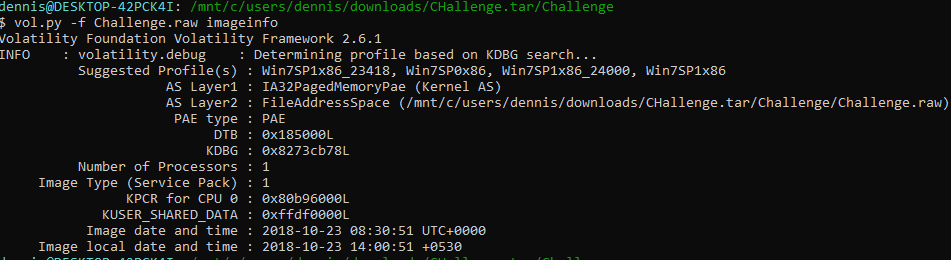
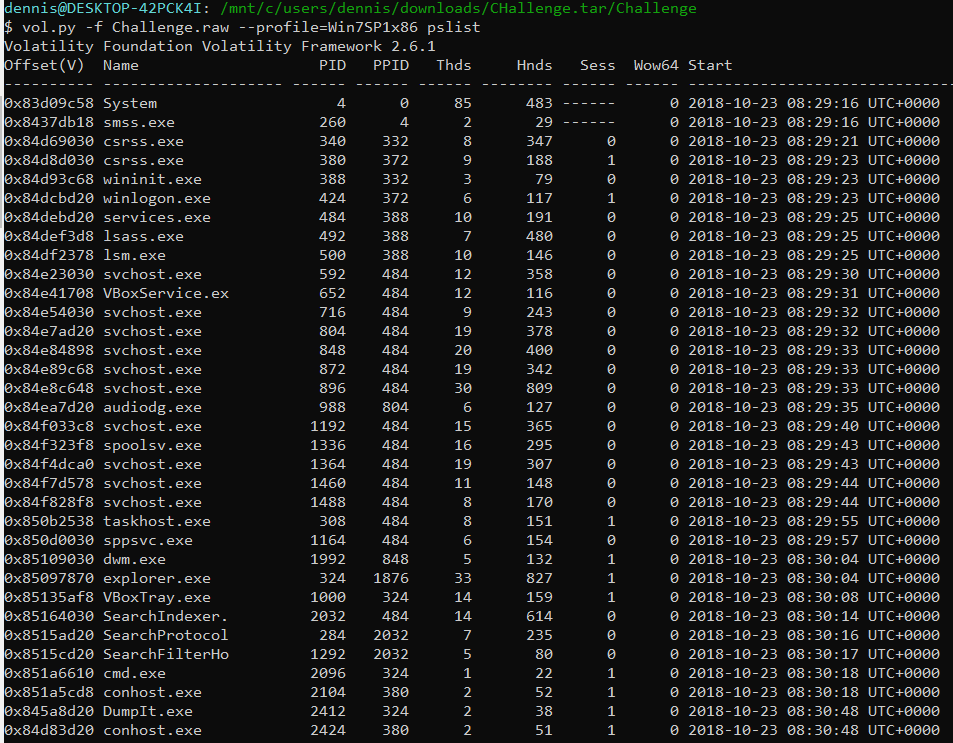
PT I - CTF challenge

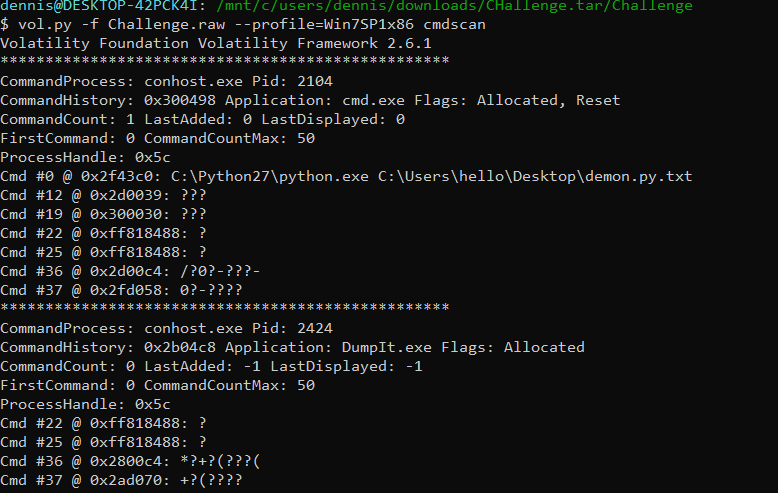
Run image info - KBDG is the kernel debugging data block that contains distinct signatures for for every Windows OS. The data block is maintained by the kernel and can provide a list of other things like running processes.



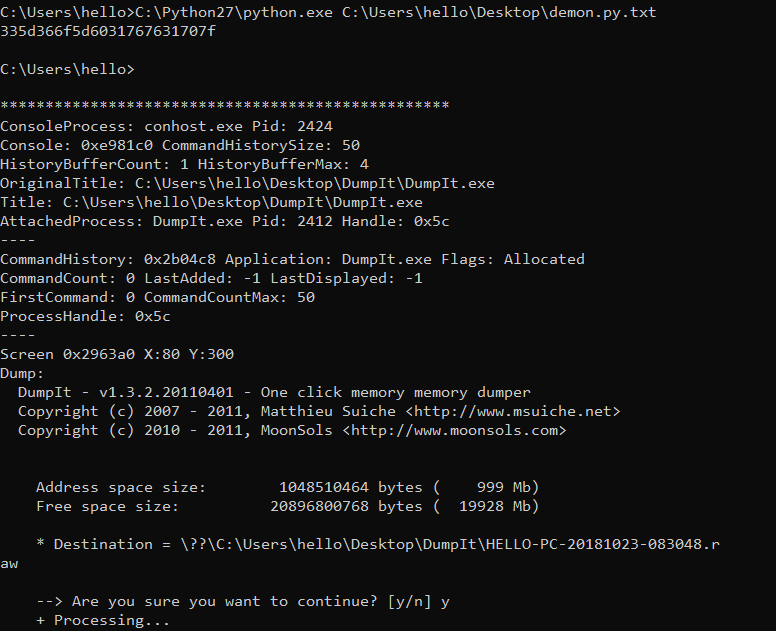
Lets run pslist to see all of the running processes



We can extract the contents of command line parameters from cmd.exe



In operating system, the standard input and standard output and the destinations where a process takes inputs and provides outputs respectively. We can look for standard outputs by providing the consoles parameter.

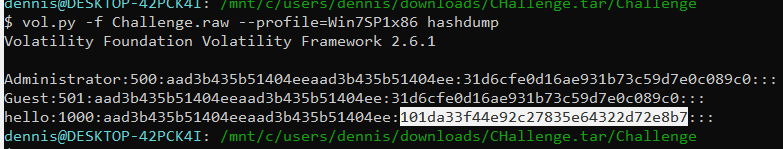


From here we get a bit string, as well as some insight into what DumpIt was (memory dumper for the lab).

We can look at environment variables using the envars plugin which results in a funny looking Variable:



We will xor the string bit and then obtain a password hash dump

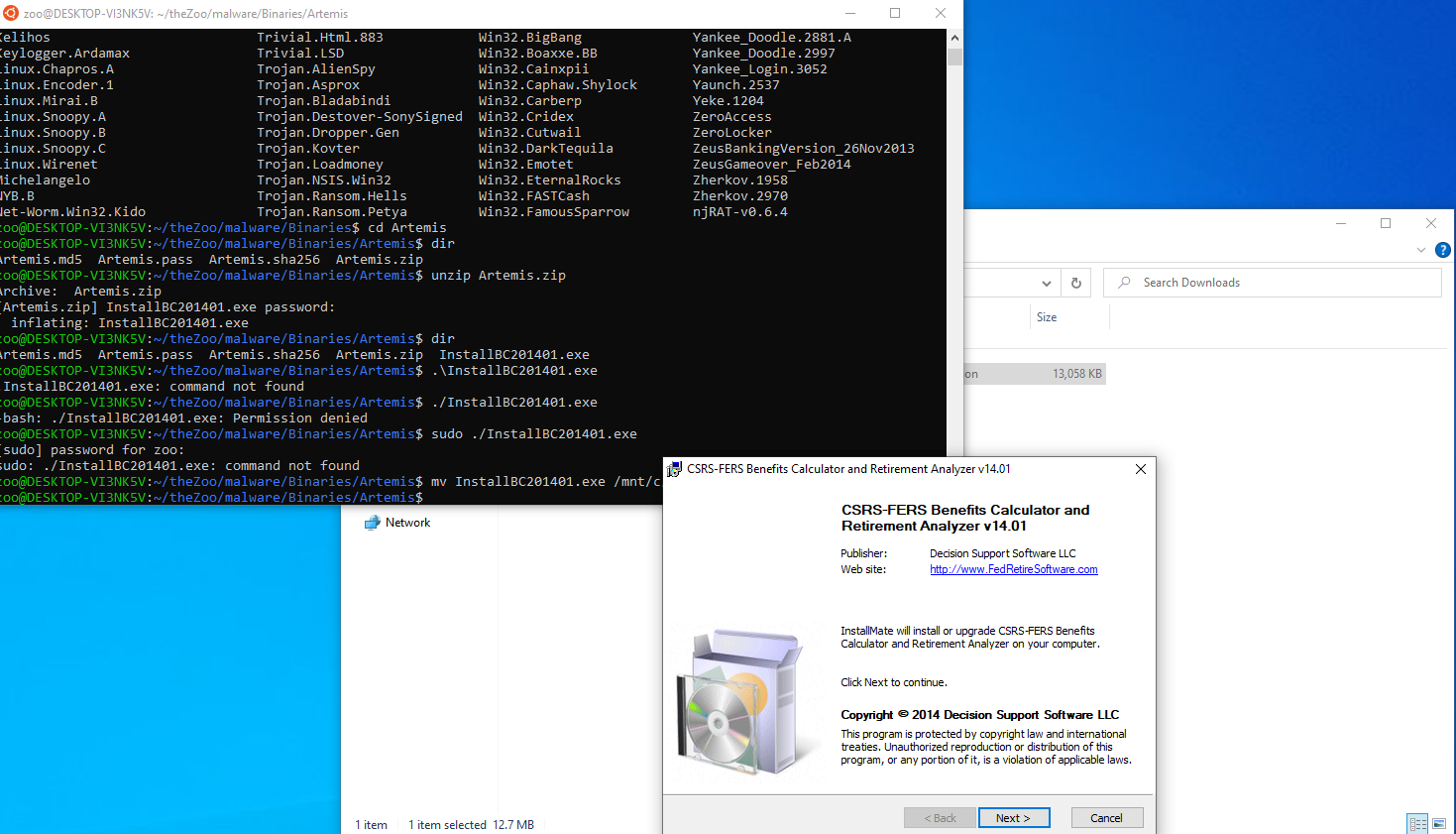


There’s a wide variety of hash databases that you can use to crack the hash:

<https://aboutdfir.com/resources/hash-lookups/>  
  
After xor and hash crack we get the flag

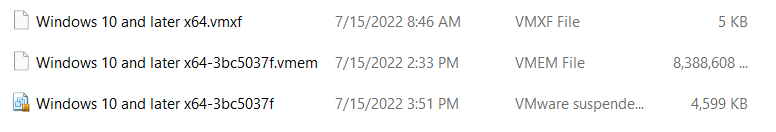
{you\_are\_good\_but1\_4m\_b3tt3r}

**PT II - Home lab**



To install TheZoo (<https://github.com/ytisf/theZoo>), you can use powershell, but I will be working on the Windows Linux Subsystem. Future endeavors include learning more about Linux forensics, and part of that journey includes using more Linux in my day to day. As a side note, TheZoo doesn't work well with Python 3 - make sure you’re using Python 2. Once TheZoo is installed, make the NIC private so traffic cannot escape the box.

I am working on a Windows 10 VMware box, so in order to capture the memory, all you need to do is suspend the VM and grab the .vmem file that gets written to disk in the VM folder.

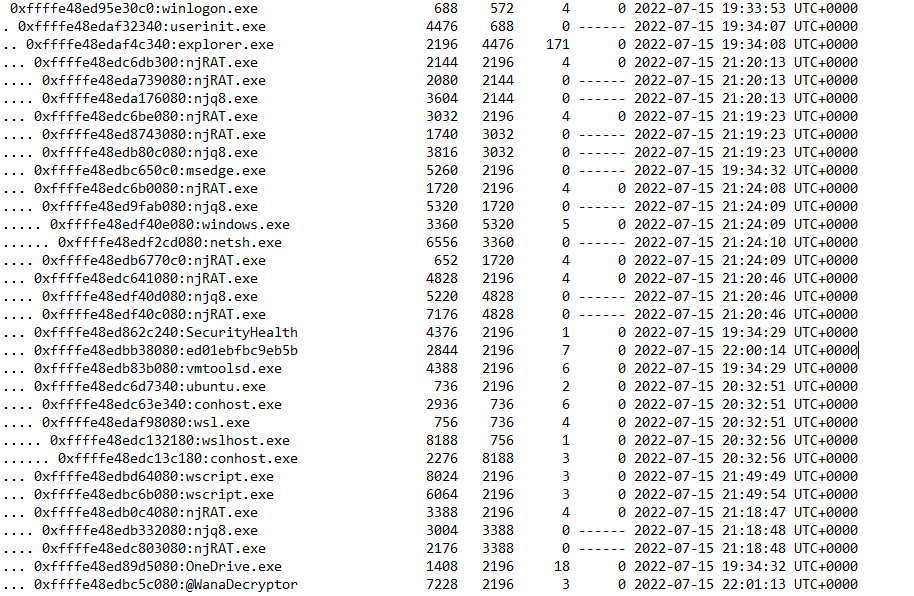


From TheZoo, cd into the Malware/Binaries folder and then cd into the malware that you want to test’s folder. Install unzip (apt-get install unzip) and unzip the malware that you want to use. We detonated a lot of malware on this device before blowing it up with WannaCry. Unfortunately, there were a lot of error messages Make a share folder in the VM settings that hosts your forensic workstation and copy the .vmem file over.   
  
Remember to turn on file sharing and network discovery in your host and vm network settings.

Next make a directory on your forensic workstation and transfer the data over from the host share to your drive.

The first thing I ran was pstree to get an info of processes and their parent relationship

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 pstree > pstree.txt

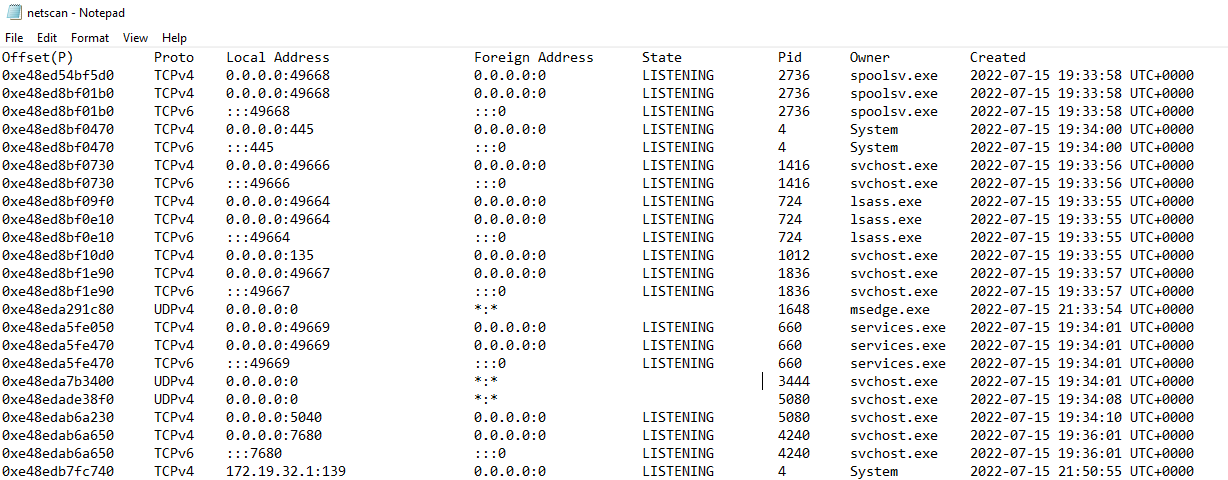


We can immediately see some of the malware successfully deployed, as well as some suspicious binaries that I’d like to have a closer look at.

* Ed01ebfbc9eb5b
* njRAT
* @WanaDecryptor

We ran netscan against the memory image and didn’t see any suspicious network connections

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 netscan > netscan.txt



If we were to find a suspicious network connection without a process name or a process ID, we can run a YARA scan against the image to obtain more information

$ vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 yarascan -U "172.19.32.1"

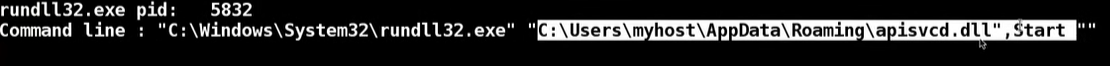
Any DLLs and processes identified can be investigated further (example from black hat proceeds)

Let’s say we found in our yara scan that Rundll32.exe was found to be connecting to a suspicious IP

We can run dlllist to see the DLLs loaded by this particular rundll32.exe

Vol.py -f xxx --profile= xx dlllist - 5832







Lets copy out the memory base address (0x00400000) and dump the DLL from memory

Vol.py -f xxx --profile= xx dlldump -p 5832 -b 0x00400000 -D /mnt/c/data

Armed with the ability to dump dlls, lets try and dump some DLLs for analysis in our lab

First run dlllist to get the dlls loaded by all processes and their corresponding CMD arguments

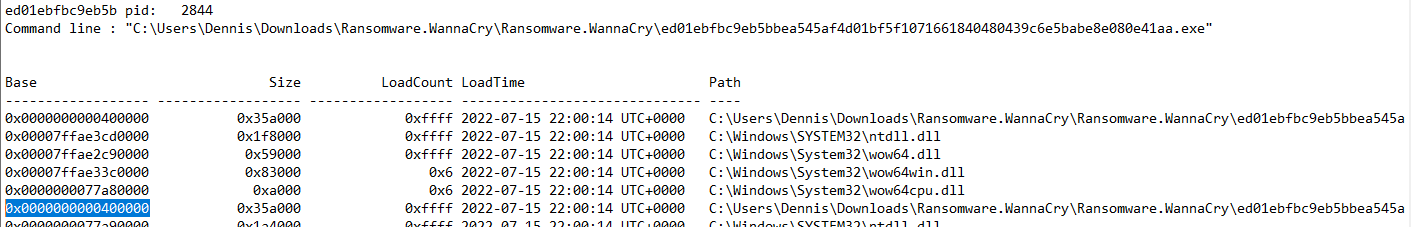
vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 dlllist > dlllist.txt

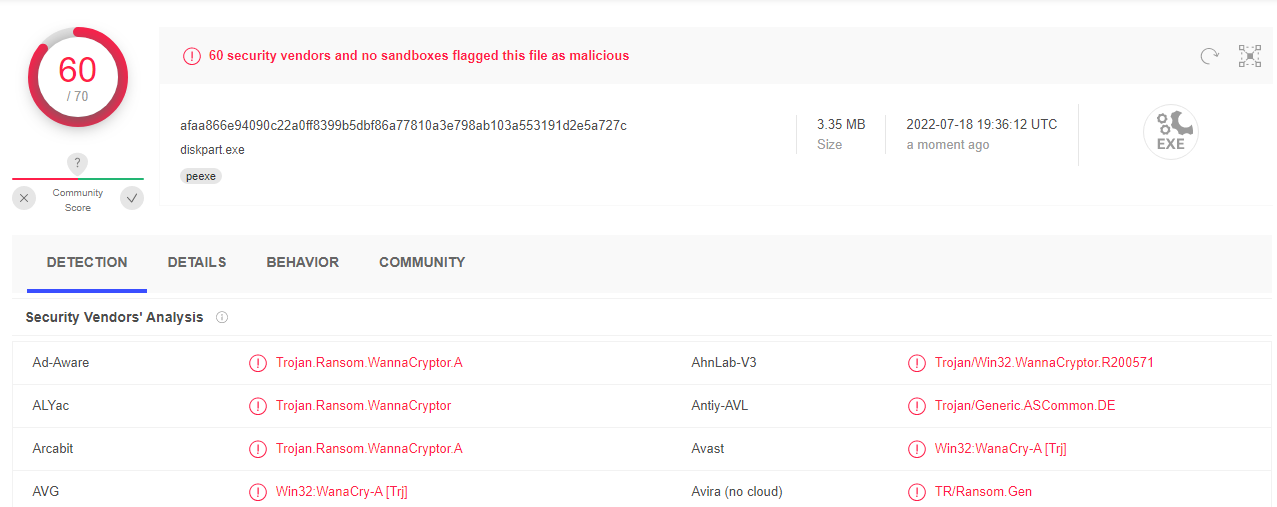


There are some WinSxS entries are interesting, which are interesting as WinSxS is a common location to put dlls if you want to [maintain persistence](https://attack.mitre.org/techniques/T1574/002/)

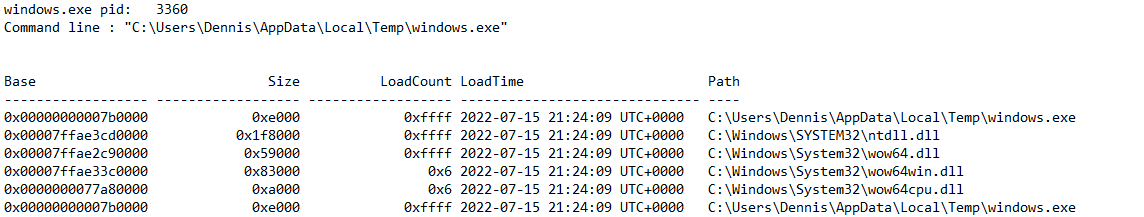


Unfortunately this dll did not populate any hits on virus total, but we were able to pull some other dlls that did light up VT





Here we see another entry in the Local Temp folder. Let’s see if we can extract that windows.exe binary:



vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 dlldump -p 3360 -b 0x00000000007b0000 -D /mnt/c/Data

Looks like we can’t extract that exe with DLL dump. I also tried a procdump

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 procdump –dumpdir /mnt/c/data

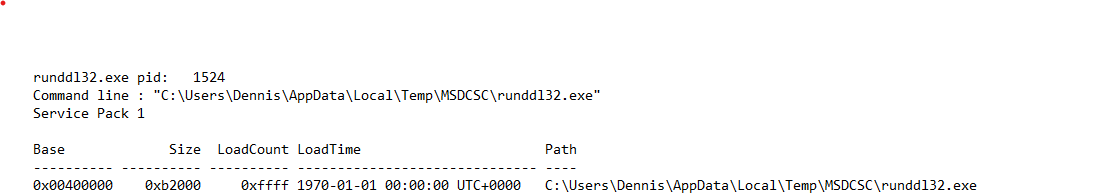
And that didn’t work either.   
  
Let’s pivot to another process



We can see runddl32.exe and a child process notepad.exe   
  
Lets run a dlllist to see where this process was being run from

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 dlllist -p 15234 > p1524dllllist.txt

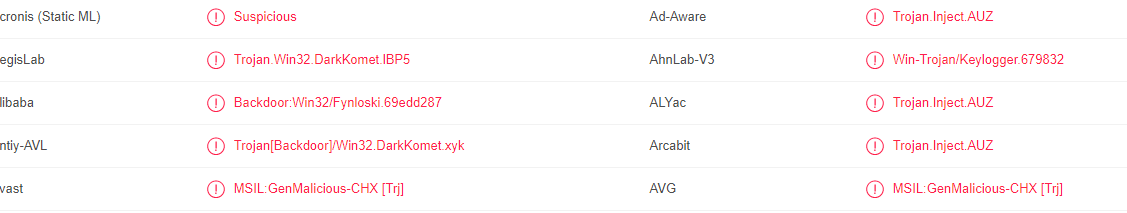
We can see that this binary is running from the Appdata\Local\Temp folder



We can use procdump to dump the file

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 procdump -D /mnt/c/data -p1524

We can upload the binary to virustotal and reveal it’s DarkComet



Another view we can use is the filescan plugin to see if the file is cached in memory

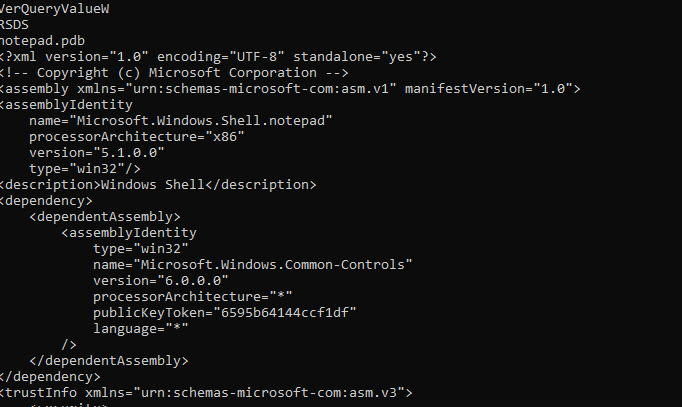
vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 filescan > filescan.txt



Now we can use the dumpfiles plugin with the -Q flag being the physical offset in memory

$ vol.py - Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 dumpfiles -D /mnt/c/data -Q 0x000000003eee8690

I also want to dump the notepad process that is the child process of runddl32.exe and use strings against it to see if I can see any information about what was written -



Looks like this is a windows shell with the public key hardcoded.

Mutex

What are Mutexes? According to [SANS](http://sans.org/blog/looking-at-mutex-objects-for-malware-discovery-indicators-of-compromise/) - Programs use mutex ("mutual exclusion") objects as a locking mechanism to serialize access to a resource on the system. How does malware muse mutex? Malware might use a mutex to avoid reinfecting the host. For instance, the specimen might attempt to open a handle to a mutex with a specific name. The specimen might exit if the mutex exists, because the host is already infected. When examining a potentially-infected system, we can look for names of mutex objects known to belong to malicious programs. The handles plugin will apply to files, registry keys, mutexes, named pipes, events, window stations, desktops, threads, and all other types of securable executive objects.

$ vol.py - f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 handles -p 1524 > handles.txt

Some malwares/exploit kits use algorithm name generations, so we can use the Mutant to not only further identify a malware, but also to create Yara rules to hunt across a network.



A quick google search for DC\_MUTEX-KHNEW06 can help us reveal that this is Darkcomet.   
  
Lets dump the files and look at them with strings -

Memdump vs Procdump

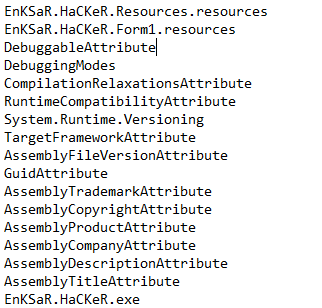
These plugins work in very different ways: dumpfiles dumps the file object as it was loaded into memory. Procdump focuses on the process executable itself. To make things a bit more complicated, plugin memdump can be used to dump every memory section corresponding to a process defined by a user. It can include loaded DLLs, file objects, network endpoint objects – anything that was loaded in the process’s memory. Running this plugin is very similar to creating a process dump on a live system, for example with SysInternals ProcDump.exe, or by rightclicking a process in Task Manager and choosing “Create a dump file” option.

Lets do a memdump on njrat

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 memdump -p 2144 -D /mnt/c/Data/Memdump/njrat

The strings output from memdump was way too much information, so lets try a procdump on njrat and wannacry and see if we can see anything interesting in the strings output.

We found some IOCs from the strings output of njrat memdump

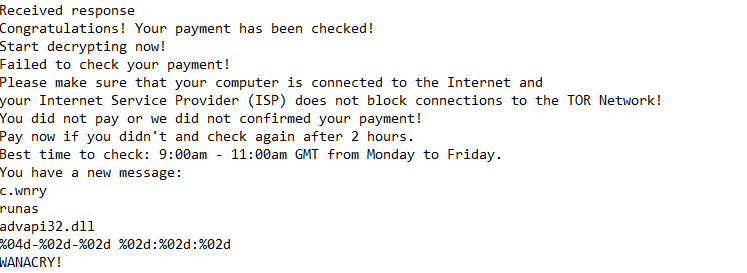


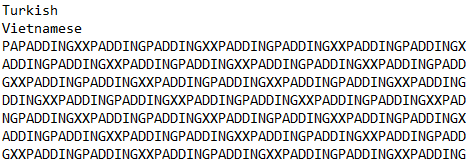
For strings in wannacry we can see some interesting indicators -

We can see where the executable deletes shadow copies



As well as some wannacry indicators

  
Finally, we can see [padding at the end of the file](https://attack.mitre.org/techniques/T1027/001/) to throw off AV



Timeline creation

Combine timeliner with mftparser and

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 mftparser -D /mnt/c/data/mftparser --output=body --output-file=mftparser.txt

vol.py -f Windows\ 10\ and\ later\ x64-3bc5037f.vmem --profile=Win10x64\_19041 timeliner --output=body --output-file=timeliner.txt

$ log2timeline.py --storage-file output.dump output.body