Malware Reverse Engineering – HW1

Problem 1: Static Analysis

Sample SHA256: 0fa1498340fca6c562cfa389ad3e93395f44c72fd128d7ba08579a69aaf3b126

First indicator from the meta data of the file: Creation date of the file is tampered. The creation date of the file says “08/30/2019” but when you check the details from virus total, It was first seen in the wild on “07/05/2011” which is way ahead of its creation date. Figure: 1 below shows the snippet from the virus total data. From the first submission date of the file to virus total, it is clear that the sample is an old file. Now, the last analysis date for it is also very recent. With this information we can take detection rate of the file in virus total as one of indicators. Since, the rate of false positives on an old file is unlikely.

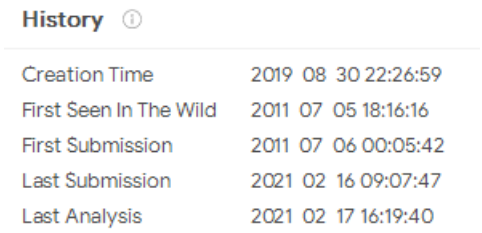


Figure 1: Basic meta data of the file

Coming to the detection rate, this sample has 59 out of 70 anti-virus companies’ verdicts as malware (Figure: 2). Some of the most reputed anti-virus companies like ESET, Kaspersky, Microsoft, McAfee, TrendMicro etc. have detected the sample as “Trojan Downloader”. This is an indicator that this sample might be used to download a malicious payload on the victim’s machine and execute. But we can’t be sure yet since we haven’t seen the IOC in the file yet. But this can be used to know what to look for in the file in the next step.

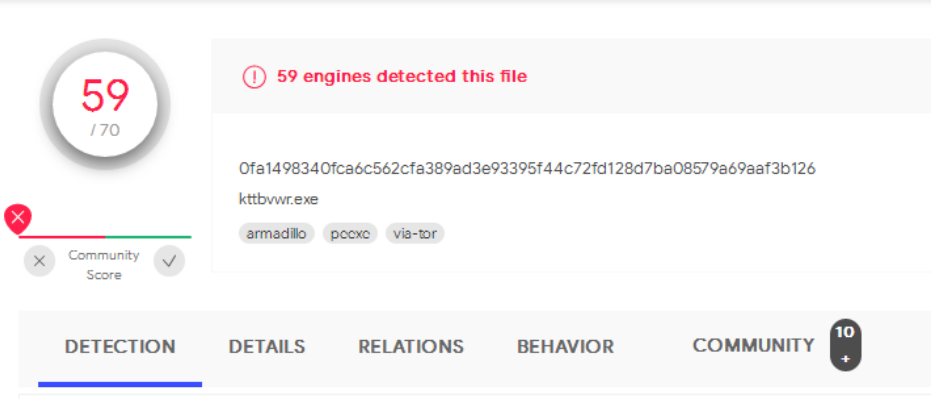


Figure 2: Detection rate

Now, we already know by looking at the virus total report that the sample is a 32-bit executable. But let’s find some other basic meta-data of the file using PEiD and PE Viewer.

From PE Viewer (Figure: 3) we can see it refers to the file as MZ header and PE header. 4D5A (MZ) is the magic number for DOS Executable and machine field hex code is ”011C” which indicates the sample is 32-bit. Looking at the hex values of the characteristics field we can see it’s “010F” which is less than “2000”. This means the sample is an executable and not dll. Image base field is “00400000” which indicates the memory address of the sample (Subjected to change once loaded into the memory, depending on the availability). Time date stamp field is “5D69A2B3” (30, August,2019) which is the compiled time of the file. Number of sections present are 4, which are .text, .rdata, .data, .rsrc. These are some commonly used sections of a PE executable and the names of which are not obfuscated. Looking at the section names, sample seems to be unpacked.

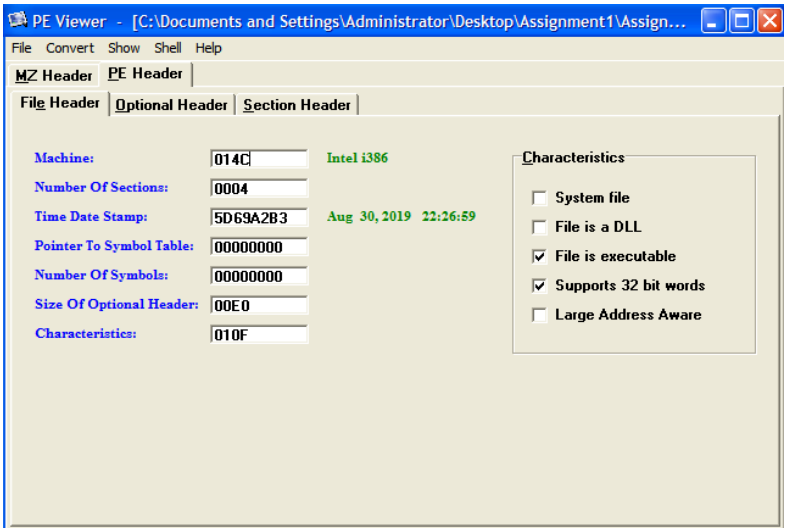


Figure 3: PE Viewer displaying PE structure of the sample

To further confirm that the sample is unpacked, we can load the file in PEiD and check for the entropy (Figure 4). If it is less than 7, the file is unpacked which in our case is another indicator that our sample is unpacked. Some other interesting data I found through PEiD is the sample is VC++ compiled and the entry point of it is in .text section.

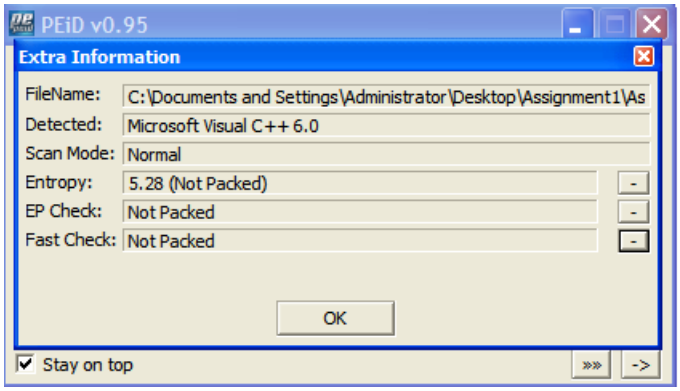


Figure 4: Entropy

Coming to seeing if the sample is packed or unpacked, one other way to check is if the strings of the sample are clear or obfuscated. We take a look at the strings from the file on disk. For doing this I'm using "Bintext" tool (Figure 5). We can see strings are not obfuscated. Hence, we can conclude that the file is not packed.

Here we also find some interesting strings. Some of them which I thought were noteworthy are: winlogon.exe, SeDebugPrivilege , \winup.exe, \system32\wupdmgrd.exe, urlmon.dll, psapi.dll, sfc_os.dll, hxxp://www[.]practicalmalwareanalysis[.]com/updater[.]exe

Winlogon.exe – It is a legitimate windows component responsible for actions at logon/logoff. But this could have been used by the sample to gain persistence by adding itself to the "userinit.exe" key of: `HKCU\Software\Microsoft\Windows NT\CurrentVersion\Winlogon\`

Like, "C:\Windows\system32\userinit.exe,sample.exe", where "C:\Windows\system32\userinit.exe" is the default data held by "userinit" key

Or it could have even used the shell key to add persistence. Like, "explorer.exe,sample.exe " where "explorer.exe" is the default data held by shell key.

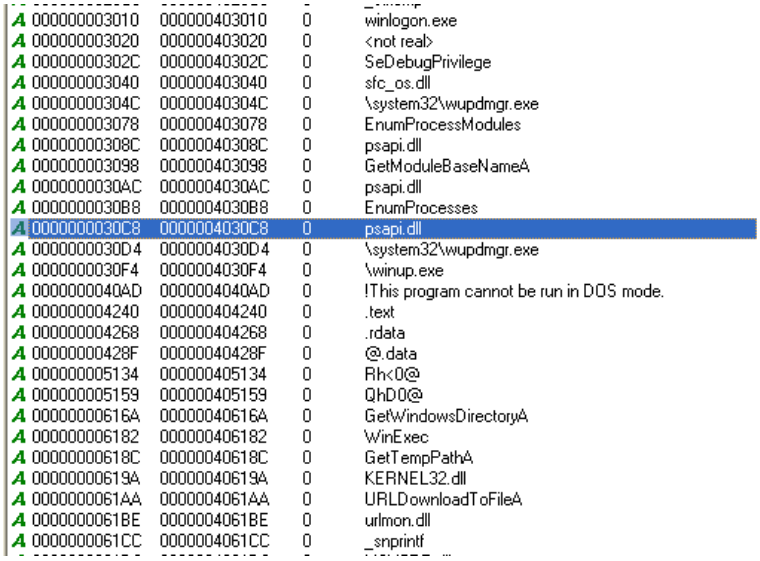
SeDebugPrivilege – Indicates that sample might be attempting to perform privilege escalation. (It is needed to interrupt a running process).

winup.exe – This could have been used by the sample to record mouse and keyboard inputs, or to manipulate other existing programs.

*system32**wupdmgrd.exe* – Path of a file in system32 directory. This might be the file whose data is overwritten by payload downloaded by this sample. While performing dynamic analysis. "\system32\wupdmgrd.exe" needs to be analyzed too. If it turns out to be performing malicious activity, it can be used as a host-based indicator to identify infected machine.

hxxp://www[.]practicalmalwareanalysis[.]com/updater[.]exe – Might be using this link to download the payload to victims' machine. It can be used as a network-based indicator to identify infected machine.

urlmon.dll, psapi.dll, sfc_os.dll – These dlls are not in the libraries imported by the sample. Sample might be doing runtime dynamic linking with these dlls to hide its purpose.



A 000000003010	000000403010	0	winlogon.exe
A 000000003020	000000403020	0	<not real>
A 00000000302C	00000040302C	0	SeDebugPrivilege
A 000000003040	000000403040	0	sfc_os.dll
A 00000000304C	00000040304C	0	\system32\wupdmgr.exe
A 000000003078	000000403078	0	EnumProcessModules
A 00000000308C	00000040308C	0	psapi.dll
A 000000003098	000000403098	0	GetModuleBaseNameA
A 0000000030AC	0000004030AC	0	psapi.dll
A 0000000030B8	0000004030B8	0	EnumProcesses
A 0000000030C8	0000004030C8	0	psapi.dll
A 0000000030D4	0000004030D4	0	\system32\wupdmgr.exe
A 0000000030F4	0000004030F4	0	\winup.exe
A 0000000040AD	0000004040AD	0	!This program cannot be run in DOS mode.
A 000000004240	000000404240	0	.text
A 000000004268	000000404268	0	.rdata
A 00000000428F	00000040428F	0	@.data
A 000000005134	000000405134	0	Rh<0@
A 000000005159	000000405159	0	QhD0@
A 00000000616A	00000040616A	0	GetWindowsDirectoryA
A 000000006182	000000406182	0	WinExec
A 00000000618C	00000040618C	0	GetTempPathA
A 00000000619A	00000040619A	0	KERNEL32.dll
A 0000000061AA	0000004061AA	0	URLDownloadToFileA
A 0000000061BE	0000004061BE	0	urlmon.dll
A 0000000061CC	0000004061CC	0	_snprintf

Figure 5: Interesting Strings

Now, we move on to checking the interesting import functions of the sample using CFF Explorer (Figure:6). Some of the functions that caught my attention are URLDownloadToFileA, GetWindowsDirectoryA, GetTempPathA, CreateFileA, WriteFile, LoadLibraryA, LookupPrivilegeValueA, CreateRemoteThread.

URLDownloadToFileA – Indicates that the sample is probably downloading file from a remote server.

GetTempPathA – Sample might be trying to access the temp folder to run a file (Since "WinExec" from strings indicates it's trying to execute a file).

CreateFileA – Sample could create a new file.

WriteFile – Sample could write a file (Can also over write an existing file when combined with other functions like *CreateRemoteThread*)

LoadLibraryA – Sample might be dynamically linking the resources.

LookupPrivilegeValueA – Might be attempting to escalate privilege value.

CreateRemoteThread – Can be used to interrupt a running process.

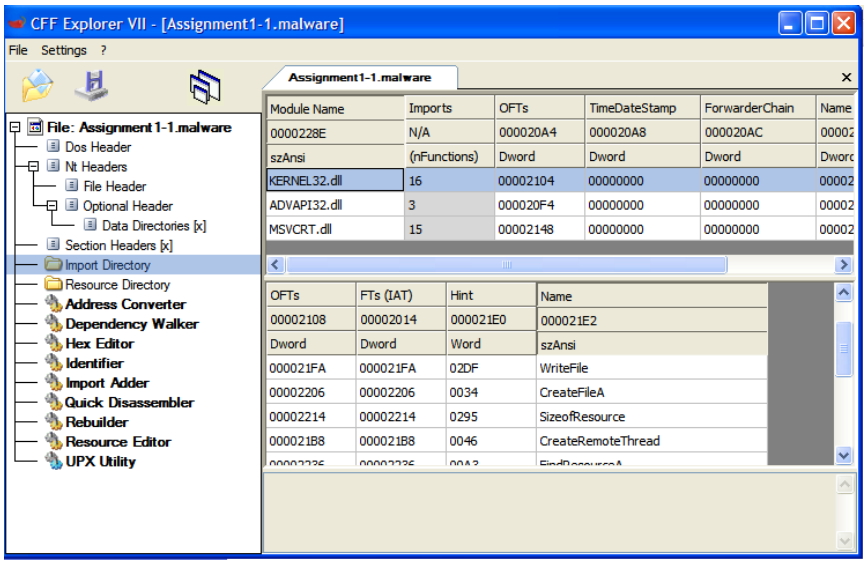


Figure 6: Interesting imports

While going through the sample in CFF Explorer, I found another interesting indicator. Sample contains another PE Executable in the resource section (Figure: 7) You can see the ascii value of the magic number is MZ and Hex value in characteristics of Image_File_Header is less than 2000. I've used Resource Hacker tool to extract the BIN file. Following are the basic meta-data of the extracted file:

SHA 256: 819b2db1876d85846811799664d512b2f1af13e329f5debe60926c3b03424745

32-bit executable

Creation time – 02/27/2011

VC++ compiled

Unpacked File.

The imports of the file contains URLDownloadToFileA, GetWindowsDirectoryA, GetTempPathA, WinExec.

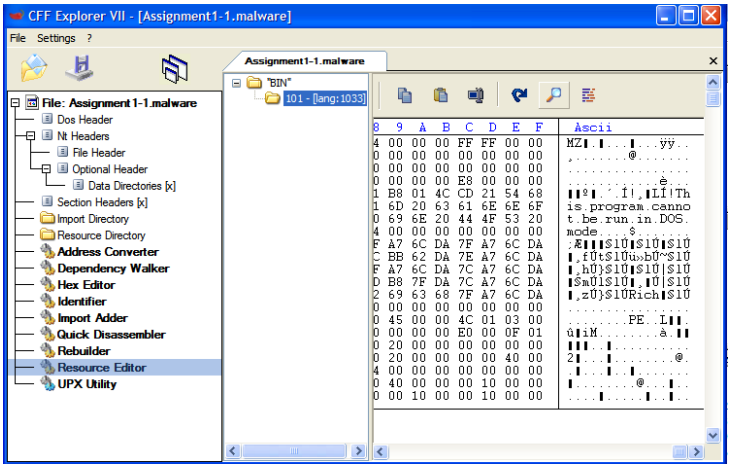


Figure 7: Executable stored in resource of the Sample

From the above observations made, the sample looks like it would drop an executable (BIN file) which then overwrites file "*system32**wupdmgrd.exe*" to mask its execution and connect to C&C server using "*hxxp://www[.]practicalmalwareanalysis[.]com/updater[.]exe"* to download malicious payload named "updater.exe". This can be confirmed when dynamic analysis is done for the sample.

Problem 2: Static Analysis

Sample SHA256: 8a35842d3f5963f715def0bbd0a53d7ffaae2d2ca79f56a5ac8bede64749d279

Questions:

1. What is the md5sum? What of interest does VirusTotal Report?

Md5sum – MD5 is one of the many hash algorithms that can be used to get a digital fingerprint of a file. The hash value generated for the file using MD5 algorithm is known as MD5 hash of the file. Md5sum is a program that can be used to calculate and verify these MD5 hashes of the files. For the given Sample, md5sum is: 02658bc9801f98dfdf167accf57f6a36

Out of many things that VirusTotal Reports, few that I find interesting are detection rate, names of the malware used by reputed anti-virus engines, first submitted date of the file, meta-data of the file, signature information, contacted URLs and IPs and import functions. For the given sample, some of the details and indicators that I can get just by reading VirusTotal Report are:

- Sample is old so the detection rate of 56/70 can be trusted and some of the trusted vendors are detecting the sample. Sample is a 32-bit executable which is VC++ compiled. Sample is not digitally signed. It fakes the name of one of the windows system processes (svchosts.exe). Contains 4 sections which are .text, .rdata, .data, .rsrc. Since the section names are clear, the sample doesn't look to be a packed file. Looking at these factors, the sample is already suspicious.

2. List a few imports or sets of imports and describe how the malware might use them.

Sleep, ReadFile, WriteFile, GetWindowsDirectoryA, InternetOpenA, InternetConnectA, HttpOpenRequestA, HttpSendRequestA, InternetWriteFile, InternetReadFile, ??3@YAXPAX@Z, ??2@YAPAXI@Z, fopen are some of the import functions that caught my attention.

??3@YAXPAX@Z, ??2@YAPAXI@Z – Indicates that the sample is masking a few import functions. A legitimate file often uses standard names.

Sleep – Sample might be using it to evade detection. This is effective when the malware is run in sandboxes. Since sandboxes have a set time to execute the file, after which it times out and stopes the execution.

fopen, ReadFile, WriteFile – Though they're very commonly used even in the legitimate file. Malware can use them to open a file to read and overwrite it. In order to hide its execution.

InternetOpenA, InternetConnectA – It opens an FTP or HTTP session for a given site. This might be used by malware to start communication over the internet.

HttpOpenRequestA, HttpSendRequestA – Malware might be trying to request a file or for any C & C instructions. Since these functions are used to create HTTP request handle.

InternetReadFile – Malware can use this to read from the handle opened by *HttpOpenRequestA* . That is, it might be reading data from an open internet file.

InternetWriteFile – Malware can use this combined with *InternetReadFile* to write a file over the internet.

From these functions we have a basic idea that the sample might be communicating over the internet to get the instructions from a C&C server and has the capability to download files from the internet.

3.What are a few strings that stick out to you and why?

Some of the strings I found interesting are: Begin Download, D-o-w-n-l-o-a-d-f-i-l-e%s******%d@@@@@@%d, Could not open file for reading, Begin Upload, U-p-l-o-a-d-f-i-l-e%s******%d, wuauclt.exe, \tasks\, cmd /c, Type command disable.Go on, svchost.exe, 69.25.50.10

Often, the error messages in the strings give out a lot of information of what the file is trying to do. Here, the sample looks like it's trying to infiltrate by downloading a file to the disc (Strings – "Begin Download, D-o-w-n-l-o-a-d-f-i-l-e%s******%d@@@@@@%d" gives of that impression) from source 69.25.50.10 and read the downloaded file (String – "Could not open file for reading") then it also looks like its trying to exfiltrate by uploading a file from the disc to a remote server data (Strings – "Begin Upload, U-p-l-o-a-d-f-i-l-e%s******%d" gives of that impression). It might be uploading the user information such as which version of the OS is being used, machine ID, IP address, 32-bit or 64-bit OS, hardware details etc. So that the attacker can send a second level payload to the victim's machine or just to check if the file is being run in a sandbox or VM. However, this is just one of the many reasons of what the attacker can do with the exfiltrated data. Now looking at the "\tasks\, cmd /c, Type command disable. Go on" string, the sample is probably trying to disable a running process through command line. "svchost.exe" string is the name of the sample. But svchost.exe is a windows system process which is digitally signed by Microsoft Corporation. And this sample has no digital signature. "wuauclt.exe" string gives off the impression that the sample might store the malicious payload in the name of the legit application "wuauclt.exe".

Problem 3: Reading assignment

Problem 4: Dynamic analysis

Sample SHA256: 8a35842d3f5963f715def0bbd0a53d7ffaae2d2ca79f56a5ac8bede64749d279

Questions:

1. What happens when you run this malware? Is it what you expected and why?

This malware has no GUI, it runs in the background for 3-4 minutes and terminates itself. This can be monitored using Process Explorer (Figure: 8). We can also see that the description and company name is same as that of svchost.exe. So, this sample is indeed hiding it's execution by pretending to be svchost.exe (like how we suspected in problem 2 while doing static analysis). One other way to verify that is to click verify under the "Image" tab of the properties when you double click on the running process on process explorer. It displays if it has a digital signature. This sample has no digital signature. We can also compare it with actual running svchost.exe and see the difference in memory strings. For this case, the memory strings from the sample were totally different from those of svchost.exe

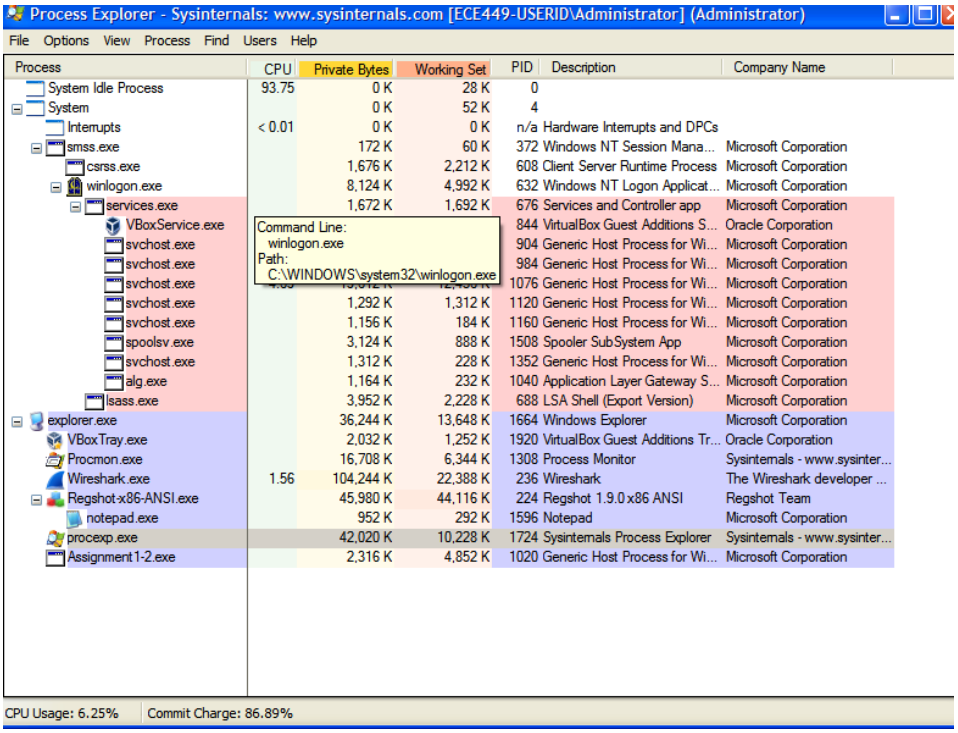


Figure 8: Monitoring process through Process explorer

Now, on further looking into process monitor, we can see in the lower panel, sample is touching windows directory in an attempt to create persistence (Figure: 9).

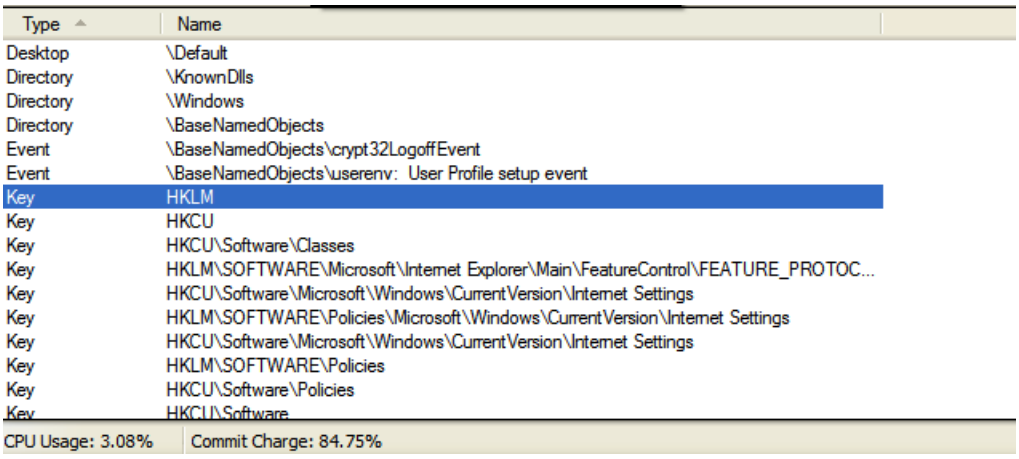


Figure 9: Accessing windows directory

Sample speeds up the boot process of itself by adding .pf copy of it in "C:\WINDOWS\Prefetch". Which is captured using regshot (Figure: 10)

```
----------------------------------
Files added: 1
----------------------------------
C:\WINDOWS\Prefetch\ASSIGNMENT1-2.EXE-1C03FEB2.pf

----------------------------------
Files [attributes?] modified: 4
----------------------------------
C:\WINDOWS\SoftwareDistribution\ReportingEvents.log
C:\WINDOWS\system32\CatRoot2\edb.chk
C:\WINDOWS\system32\config\software.LOG
C:\WINDOWS\WindowsUpdate.log

----------------------------------
```

Figure 10: Adding itself to prefetched files by windows

Sample also modified 4 files on disk as shown in figure: 10. Now coming to the network activity, there are attempts of communications between the host and IP "69.25.50.10" (Figure: 11). But the TCP connection has not been established and the communication is out-of-order.

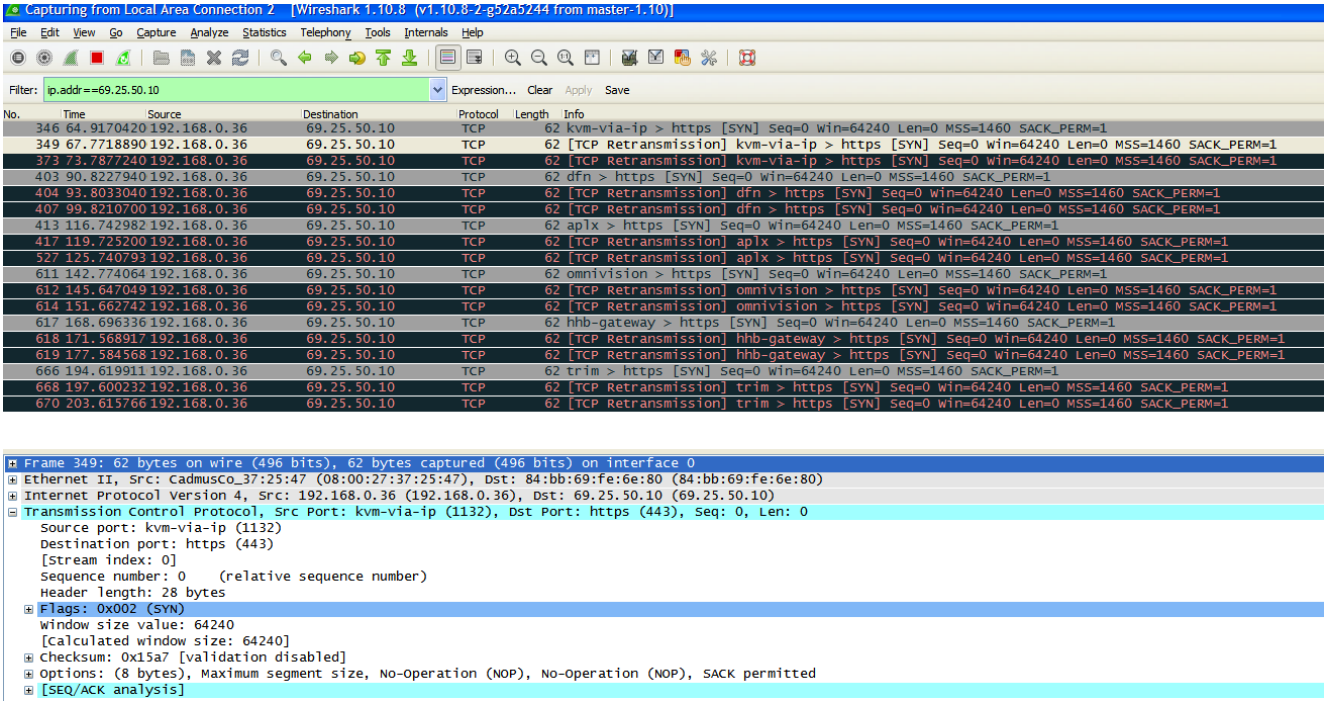


Figure 11: Network connection attempt

Looking at this host and network activity of the sample, it looks like this malware doesn't and show all the malicious signs we predicted from static analysis and closes itself after running in the background. Why? It might be having a pre-check before showing its actual behavior. According to me, sample is checking for the presence of VM and since the condition for it satisfies, it is terminated without leaving many footprints.

2. Name a procmon filter and why you used it.

Filter used – "Path" "is"

"C:\DocumentsandSettings\Administrator\Desktop\Assignment1\Assignment1\Assignment1-2.malware"

Giving the location of the Sample file as filter is one of the easiest ways to get all the operations done on the sample. It displayed only one operation (Figure: 12) "CloseFile" which is used to close the running file. This helped me make sense of why here, when I filtered using the Process Name, nothing was displayed. It was cause the sample itself dint perform the host-based activities until it satisfied the condition that it was not running on VM. Since in my case that condition is not satisfied, the program is closed.

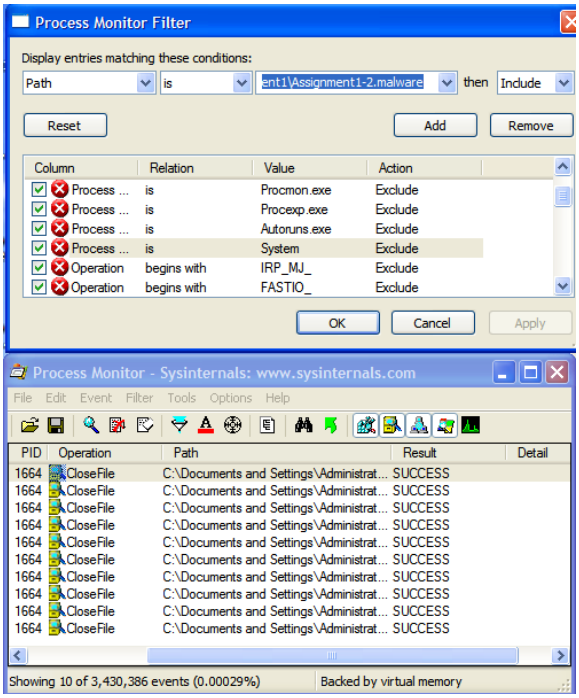


Figure 12: ProcMon filter

3.Are there any host-based signatures? (Files, registry keys, processes or services, etc). If so, what are they?

Yes, we can use the combination of the following to write host-based signatures:

D-o-w-n-l-o-a-d-f-i-l-e%s******%d@@@@@@%d

U-p-l-o-a-d-f-i-l-e%s******%d

wuauclt.exe

\tasks\

cmd /c

C:\WINDOWS\Prefetch\ASSIGNMENT1-2.EXE-1C03FEB2.pf

HKU\S-1-5-21-329068152-1897051121-1801674531-500\Software\Microsoft\Windows\ShellNoRoam\MUICache\C:\DocumentsandSettings\Administrator\Desktop\Assignment1\Assignment1\Assignment1-2.exe: "Generic Host Process for Win32 Services"

C:\ProgramFiles\Google\Update\Install\{A6F839BB-E633-4F5A-ADF3-810431150145}\69.0.3497.100_68.0.3440.106_chrome_updater.exe" --verbose-logging --do-not-launch-chrome --system-level

4. Are there any network-based signatures? (URLs, packet contents. etc) If so, what are they?

Yes, following can be used to write a network-based signature:

hxxps://69[.]25[.]50[.]10/

hxxp://69[.]25[.]50[.]10/

5. Is there anything that impeded your analysis? How so? How might you overcome this?

Yes, the sample did not show its full behavior it intended to. One reason is it checked the version info (string: VS_VERSION_INFO) of my OS and decided to not perform its activity in windows xP since most of the researchers use windows xP for analysis. Another reason is it checked for the condition to see if it was executed in a virtual machine and got terminated.

But we can overcome this, one way of removing the evasion techniques here is by replacing the registers and the lines of logics used for the check, in assembly language with NOP values. Saving the changes and executing the saved file will show us the actual intention of the sample.

6.What do you think is the purpose of this malware?

From all the observations made above, this malware looks like a trojan which communicates over the network to exfiltrate data from the victim to C&C server after infiltrating it with shell commands as the malicious payload.

References:

[1] https://docs.microsoft.com/en-us/windows/win32/api/

[2] https://www.virustotal.com/

[3] https://www.lastline.com/blog/evasive-malware-detects-and-defeats-virtual-machine-analysis/