CS4238 Report

Project 2 Team 4

Team Member	Matriculation No.	Email
HAZIQ HAKIM BIN ABDUL RAHMAN	A0216481H	e0540038@u.nus.edu
NG JONG RAY, EDWARD	A0216695U	e0540252@u.nus.edu
THIO LENG KIAT	A0233462L	e0725462@u.nus.edu

Everyone contributed equally

Static Analysis

VirusTotal:

The sample is flagged by VirusTotal as malicious with a detection score of 54/71. (Fig 1)

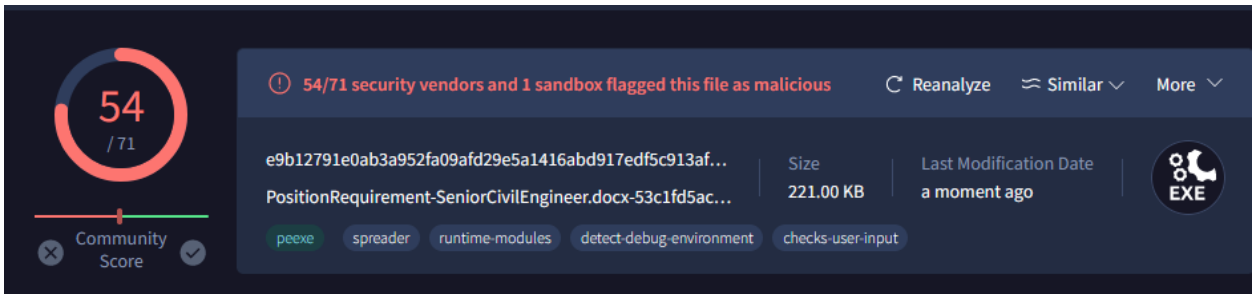

Fig 1: VirusTotal score for the malware sample

Moreover, Virustotal shows that the program has the evasive capability to detect if it is running in a debugger (Fig 2). Hence, there is a possibility that the behaviour of the malware sample could change when running in a debugger.

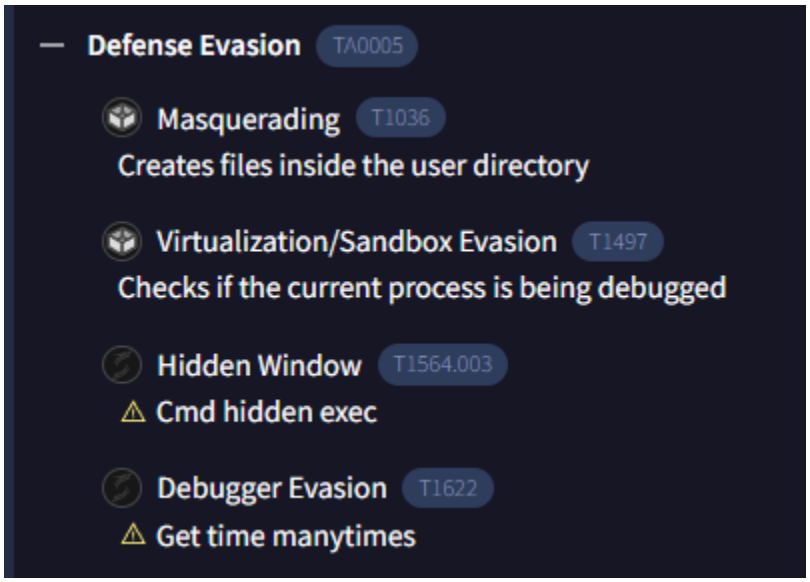

Fig 2: Malware capabilities by VirusTotal

PEView:

We first found the compilation time (Fig 3). This corresponds with the timeline of the SingHealth Data breach which started between 2017 August to 2018 July.

pFile	Data	Description	Value
000000E4	014C	Machine	IMAGE_FILE_MACHINE_I386
000000E6	0005	Number of Sections	
000000E8	5AD41B92	Time Date Stamp	2018/04/16 Mon 03:42:10 UTC
000000EC	00000000	Pointer to Symbol Table	
000000F0	00000000	Number of Symbols	

Fig 3: Using PEView to find the compilation time of the sample

PEView also shows the windows functions and dlls imported (Fig 4).

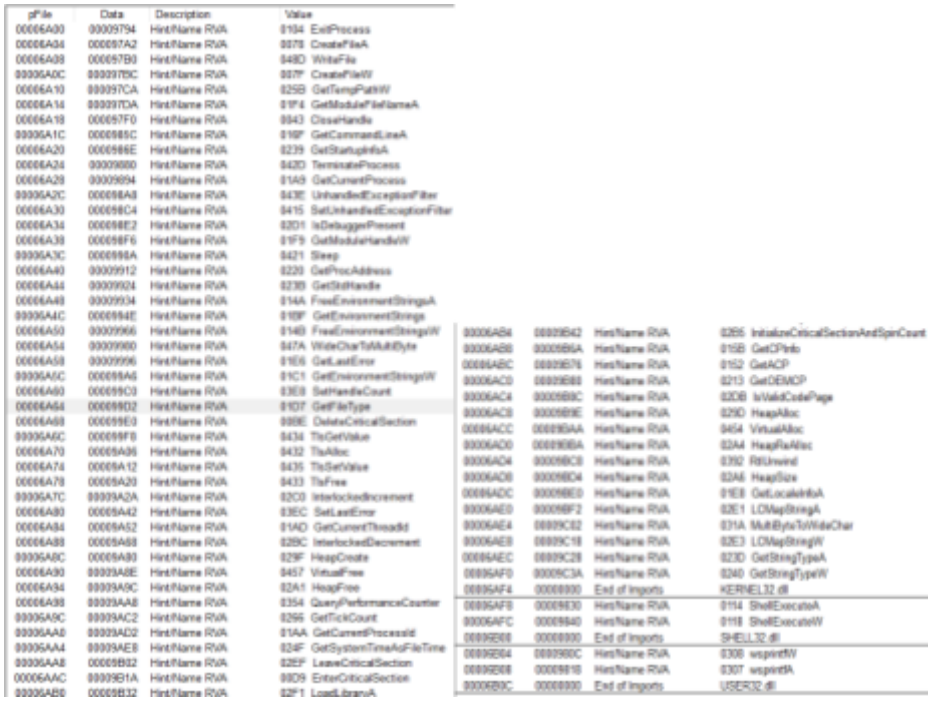

Fig 4: Imported functions and libraries by the malware sample

`ShellExecuteA` and `ShellExecuteW` are functions used to execute a specified file or program. `CreateFileA` and `CreateFileW` can be used to create or open existing files. `WriteFile` writes data to a file. These functions can allow the malware to write and alter data to then execute them.

`DeleteCriticalSection`, `EnterCriticalSection`, `LeaveCriticalSection` and `InitializeCriticalSectionAndSpinCount`. These functions allow the malware to control the flow of execution and or to manipulate threads.

`GetTempPathW` is a function that returns the path of the system's temporary folder. This could indicate that the malware could hide files or executables in the temp folder.

Strings:

Running `strings` on the malware returns multiple strings which could be of interest. This includes: `adobe.exe`, `MSVCR100.dll`, `MSVCR110.dat`, `eeclnt.exe` and `a.bat` batch script file (Fig 5).

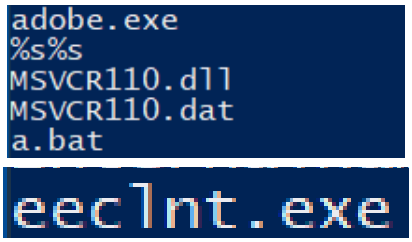

Fig 5: Filenames found in strings

Moreover, additional strings are found after the `a.bat` file (Fig 6). This could potentially contain the code for the `a.bat`. `%s` could indicate a string variable and `del` command deletes a file in the current folder with the name in the `%s` variable. Hence, `%s` could potentially be the filename of the original malware sample as it can delete the malware sample to hide its tracks.

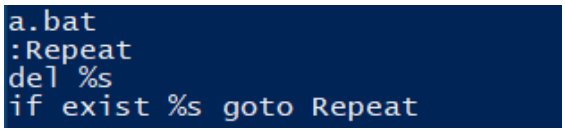

Fig 6: `a.bat` batch script

Certificate signing strings have also been discovered. Verisign Inc. is a company that operates a diverse array of network infrastructure, including two of the Internet's thirteen root name servers (Fig 7).

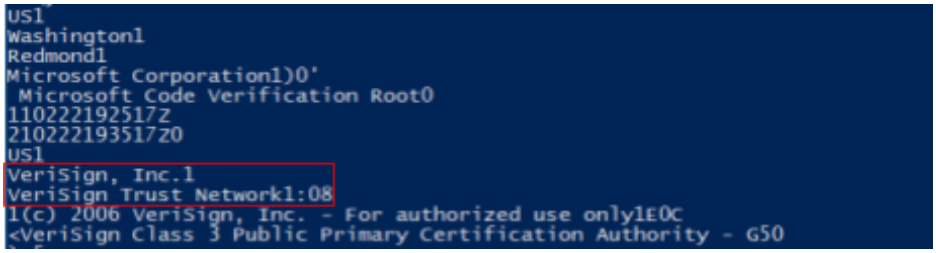

Fig 7: VeriSign string

Thawte is a company acting as a certificate authority for X.509 certificates (Fig 8).

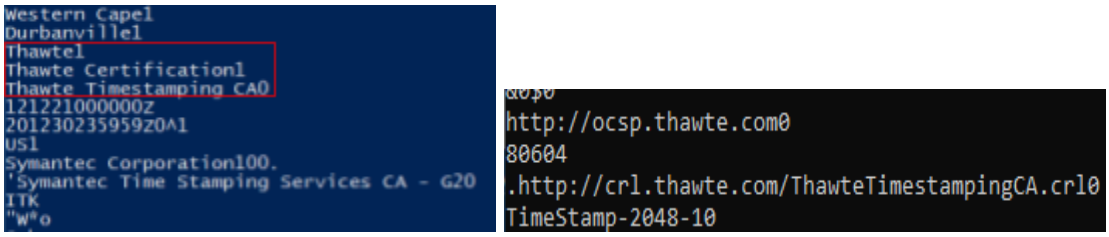

Fig 8: Thawte string

Symantec is a security solution company that provides timestamping services (Fig 9). Timestamping services could be used by the attacker to sign executables, dlls or other files and programs created by them so that the malware created appears legitimate.

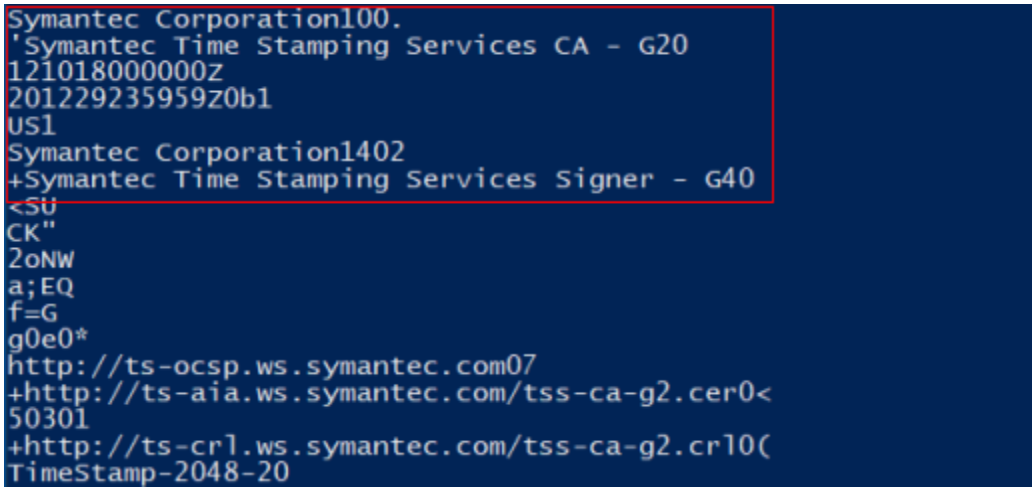

Fig 9: Symantec Time Stamping Service String

ESET is a cybersecurity software company. ESET Smart Security is an antivirus from ESET (Fig 10).

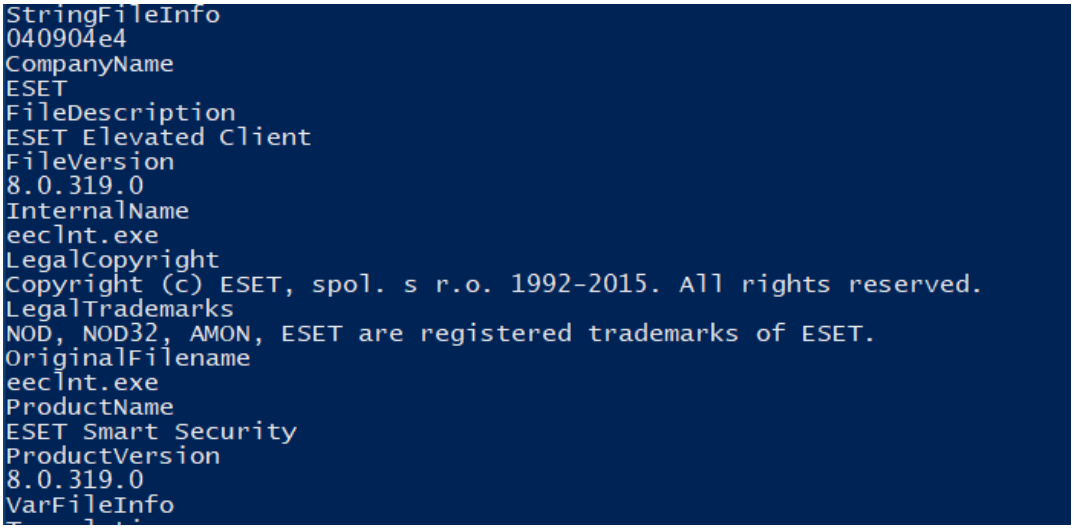

Fig 10: ESET Smart Security String

`advapi32.dll` supports security and registry creation, edits among other functions. `psapi.dll` makes it easier to obtain information about processes and device drivers (Fig 11). These two dll files are interesting as they were not picked up in the PEView.

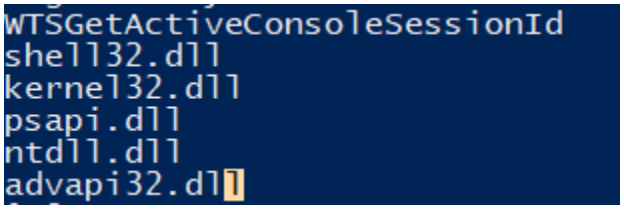

Fig 11: advapi32.dll and psapi.dll found

Resource Hacker:

Moreover, using Resource Hacker, the screenshot below (Fig 12) seems to be the thumbnail logo that accompanies the malware. Even though the malware sample is an `.exe` extension, it will show the microsoft word logo instead of the executable logo. This could indicate that the malware is a type of trojan, to trick the victim into thinking that it's a word document and hence running the malware executable.

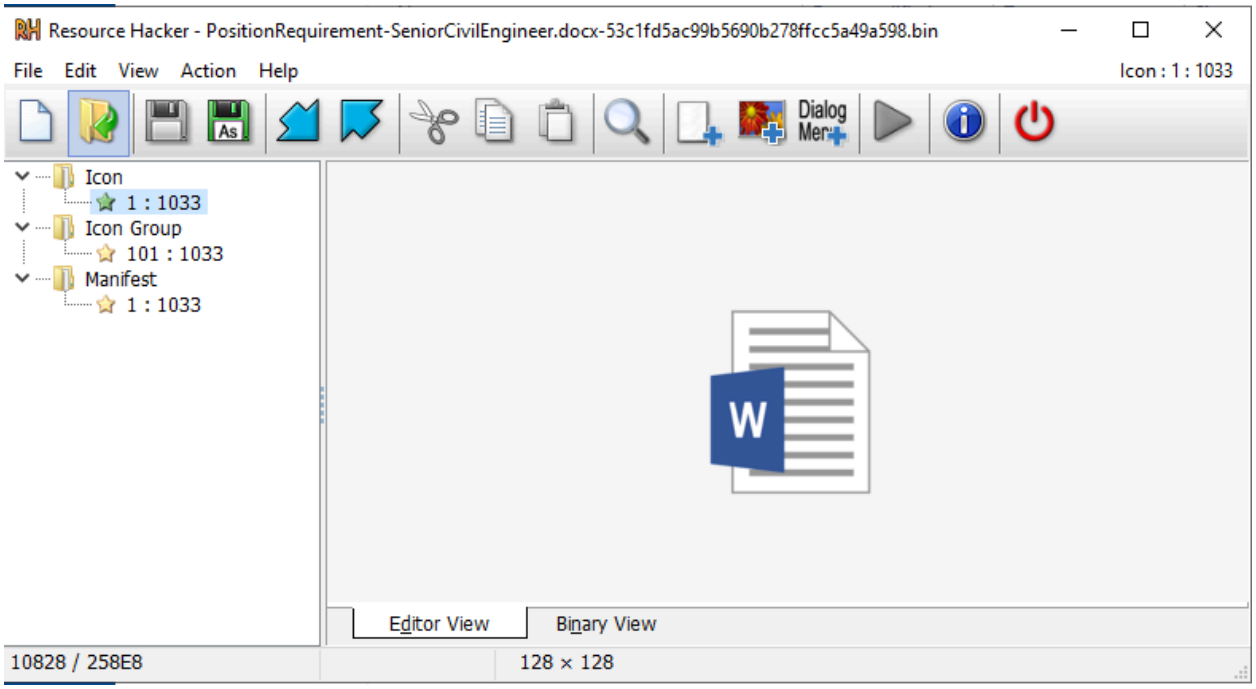

Fig 12: Word Document icon resource found in the malware sample

IDA:

IDA is a disassembler which allows us to disassemble the malware sample. Through IDA Pro, we can find in the subroutine `sub_401000` corroborates the strings command where the malware creates the required malicious programs (Fig 13). The `MSVCR110.dll`, `MSVCR110.dat` and `a.bat` file are created by the malware.

```
mov     [esp+1074h+var_A04], dx
call    sub_405730
add     esp, 3Ch
lea     ecx, [esp+1038h+Buffer]
push    ecx             ; lpBuffer
push    104h            ; nBufferLength
call    ds:GetTempPathW
mov     esi, ds:wsprintfW
push    offset aAdobeExe ; "adobe.exe"
lea     edx, [esp+103Ch+Buffer]
push    edx
lea     eax, [esp+1040h+File]
push    offset aSS      ; "%s%s"
push    eax             ; LPWSTR
call    esi ; wsprintfW
push    offset aMsvcr110Dll ; "MSVCR110.dll"
lea     ecx, [esp+104Ch+Buffer]
push    ecx
lea     edx, [esp+1050h+var_5F4]
push    offset aSS      ; "%s%s"
push    edx             ; LPWSTR
call    esi ; wsprintfW
push    offset aMsvcr110Dat ; "MSVCR110.dat"
lea     eax, [esp+105Ch+Buffer]
push    eax
lea     ecx, [esp+1060h+FileName]
push    offset aSS      ; "%s%s"
push    ecx             ; LPWSTR
call    esi ; wsprintfW
push    offset aABat    ; "a.bat"
lea     edx, [esp+106Ch+Buffer]
push    edx
lea     eax, [esp+1070h+var_A04]
push    offset aSS      ; "%s%s"
push    eax             ; LPWSTR
call    esi ; wsprintfW
add     esp, 40h
```

```
mov     esi, ds:CreateFileW
lea     ecx, [esp+1050h+FileName]
push    ecx             ; lpFileName
call    esi ; CreateFileW
push    0               ; hTemplateFile
push    80h             ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    2               ; dwShareMode
push    40000000h       ; dwDesiredAccess
lea     edx, [esp+1050h+File]
push    edx             ; lpFileName
mov     edi, eax
call    esi ; CreateFileW
push    0               ; hTemplateFile
push    80h             ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    2               ; dwShareMode
mov     ebp, eax
push    40000000h       ; dwDesiredAccess
lea     eax, [esp+1050h+var_5F4]
push    eax             ; lpFileName
mov     [esp+1054h+hFile], ebp
call    esi ; CreateFileW
push    0               ; hTemplateFile
push    80h             ; dwFlagsAndAttributes
push    2               ; dwCreationDisposition
push    0               ; lpSecurityAttributes
push    2               ; dwShareMode
push    40000000h       ; dwDesiredAccess
lea     ecx, [esp+1050h+var_A04]
push    ecx             ; lpFileName
```

Fig 13: sub_401000 showing the strings of the files created

Moreover, subroutine `sub_401000` also shows the malware sample calling `ShellExecuteW` with the "open" operation (Fig 14). This could indicate the malware sample launching `adobe.exe`

Fig 14: sub_401000 calling ShellExecuteA

Moreover, there seems to be a check in the malware sample to detect if it is in a debugging environment by the function isDebuggerPresent (Fig 15). If present, the malware will likely just terminate.

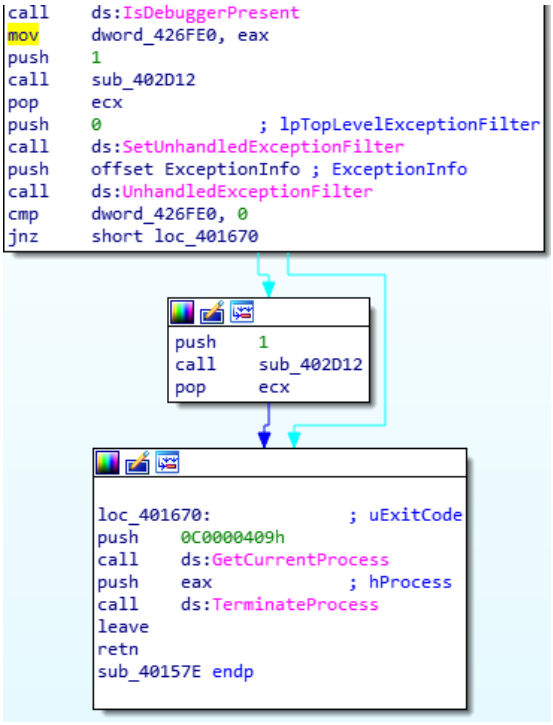

Fig 15: Malware sample checking if it is in a debugging environment

Dynamic Analysis

When changing the file extension to `.exe`, we notice that the file changes its icon as predicted during the static analysis (Fig 16).

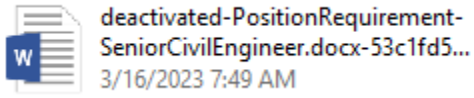

Fig 16: Exe file changes thumbnail icon to a word doc

When running the malware sample, the command prompt seemed to open briefly, indicating that some commands were executed by the malware, subsequently opening a docx file (Fig 17). This corroborates with our initial impression that this trojan malware attempts to trick users into clicking it.

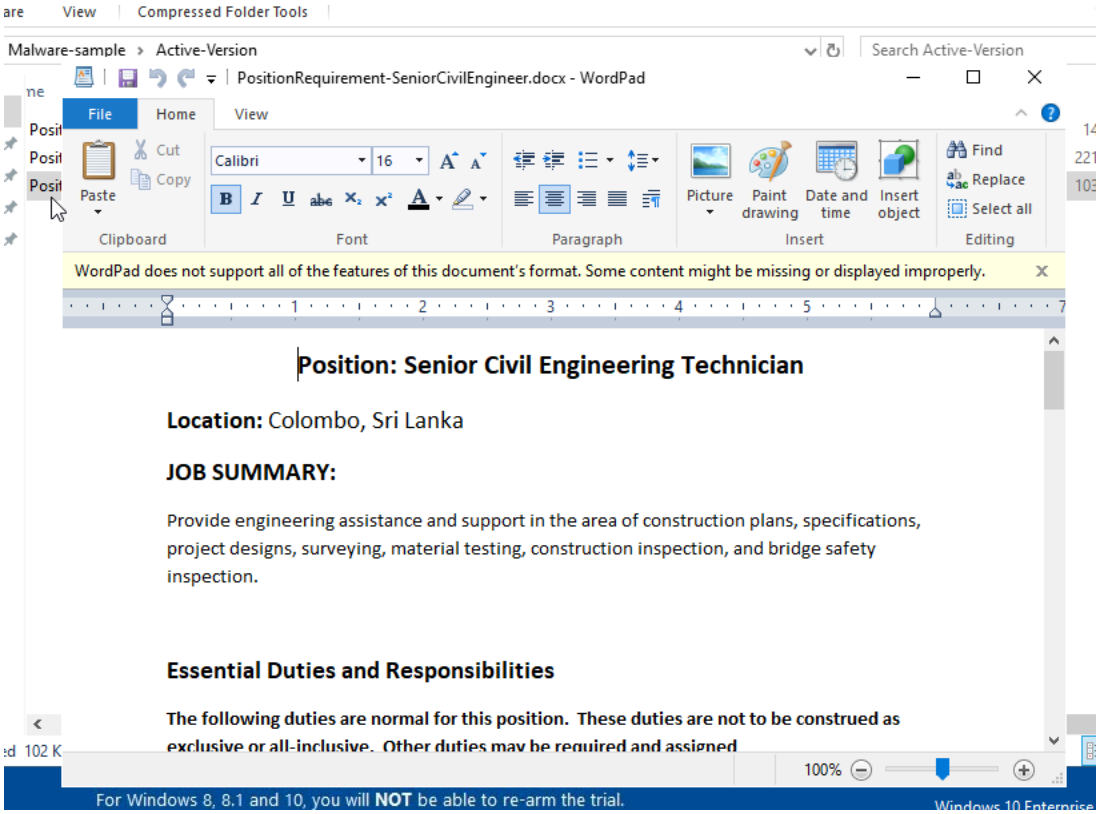

Fig 17: Word Doc that opens after running the malware

Moreover, it seems that malware will replace itself with the legitimate docs file after executing, indicating some type of evasive manoeuvre. This can be seen from the decrease in size of the file (Fig 18).

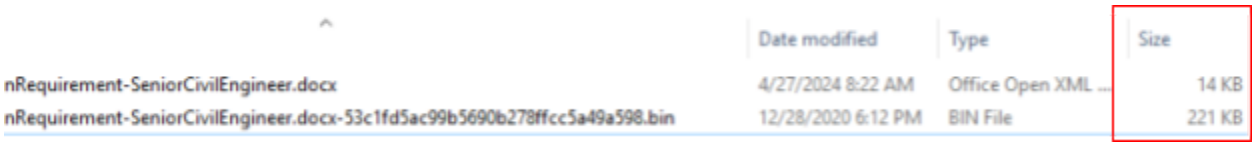

Fig 18: File size decrease

From the previous static analysis, we noted that the malware created `adobe.exe`, `MSVCR110.dll`, `MSVCR110.dat`, `eeclnt.exe` and `a.bat` as well as the knowledge that `GetTempPathW` function was imported, the Temp directory was the first place to check for any changes. The Temp directory did contain the files which we expected to see (Fig 19).

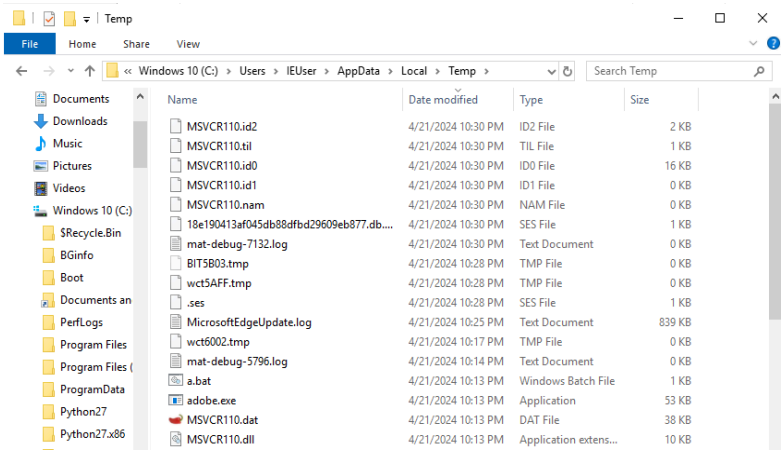

Fig 19: Temp folder containing the files which the malware created

The adobe executable is signed by ESET, hence it could be seemingly legitimate (Fig 20). This corresponds with the signing authorities as shown in the strings command previously.

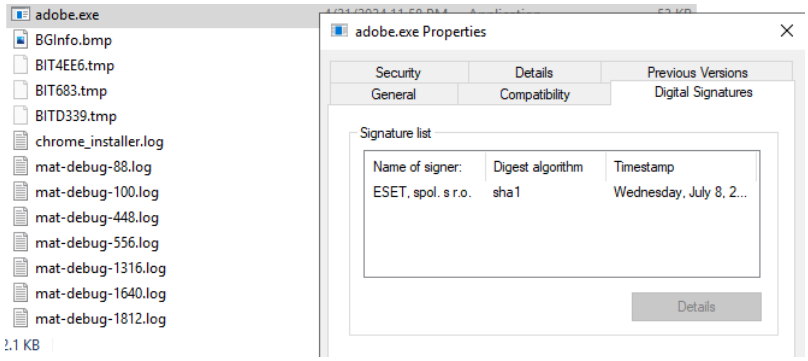

Fig 20: Signed adobe.exe by ESET

Another observation from Fig 19 is that `MSVCR110.dll` is in the same directory as `adobe.exe`. This could indicate that the malware utilises DLL side loading. This allows a legitimate executable such as `adobe.exe` to load a malicious `MSVCR110.dll` instead of the legitimate `MSVCR110.dll` because Windows will locate the required dll from the current directory before it checks other system folders.

Opening `a.bat`, we see that it is a small batch script to delete the malware once it has been executed (Fig 21). This should explain the previous observation why the file size suddenly decreases (Fig 18).

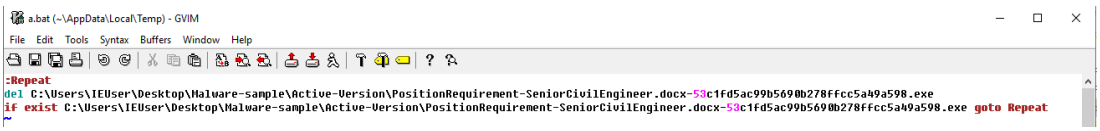

Fig 21: `a.bat` script

Moreover, we noticed that `eeclnt.exe`, `MSVCR110.dll` and `MSVCR110.dat` can be found in this folder `C:\Users\IEUser\AppData\Roaming\Windows`. This folder has been configured as HIDDEN | SYSTEM with unrestricted SYSTEM access hence could be a reason why the files were left here as well (Fig 22). Moreover, an observation is that `eeclnt.exe` could likely be a file by ESET Smart Security.

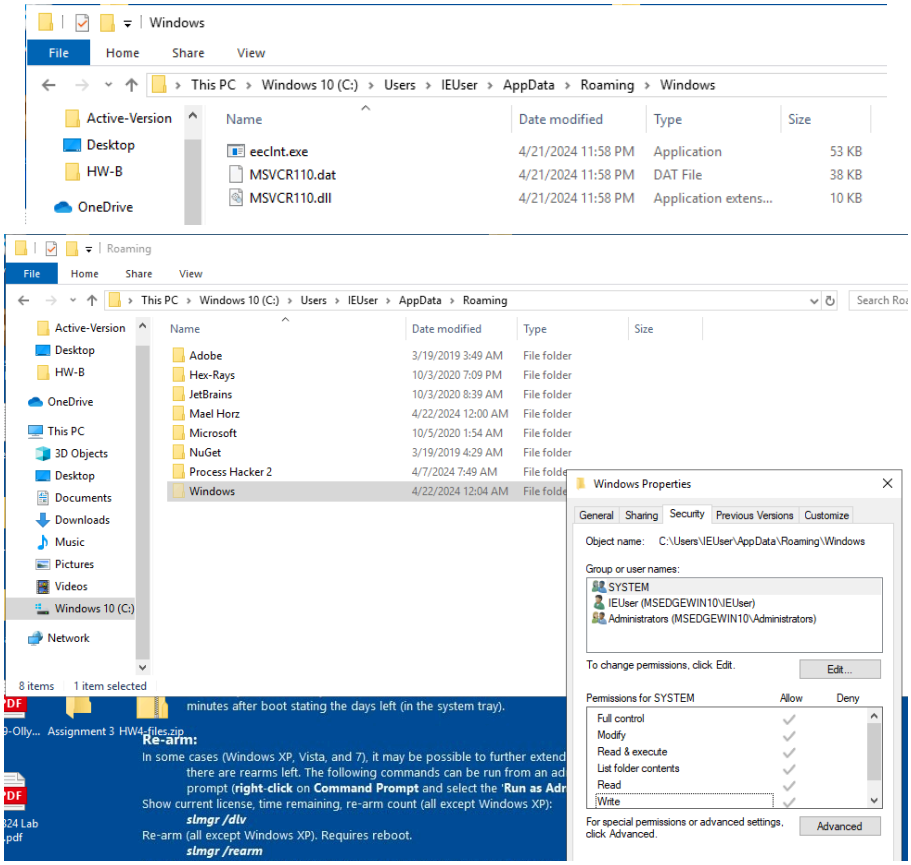

Fig 22: Files created by malware in AppData\Roaming\Windows

Moreover, eeclnt.exe also creates a new service "WanServer" with an extremely vague description of its purpose (Fig 23). Moreover, it seems that this service is started with a command line argument "260" when executing `eeclnt.exe`. This could act as a form of persistence, as victims might assume this service is important to the computer's networking functions

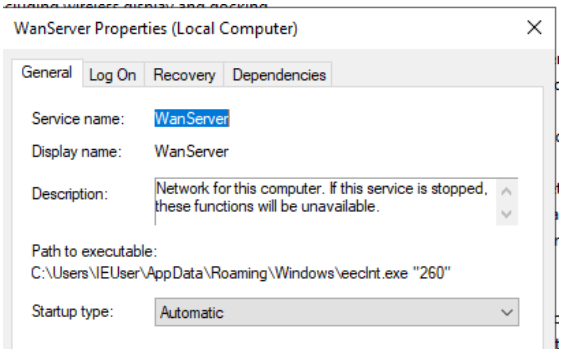

Fig 23: WanServer created

Comparing the hashes between the different folders, it is clear that `eeclnt.exe` and `adobe.exe` have the same hash values for both SHA-256 and MD5 (Fig 24). This highly suggests that `eeclnt.exe` and `adobe.exe` are the same executables. This could disprove the previous assumption that `eeclnt.exe` and `adobe.exe` are legitimate executables because the authentic executables should have different functions and hence produce different hashes. This gives the impression that `eeclnt.exe` acts as an additional form of persistence that is placed in a hidden folder should the victim discover adobe.exe.

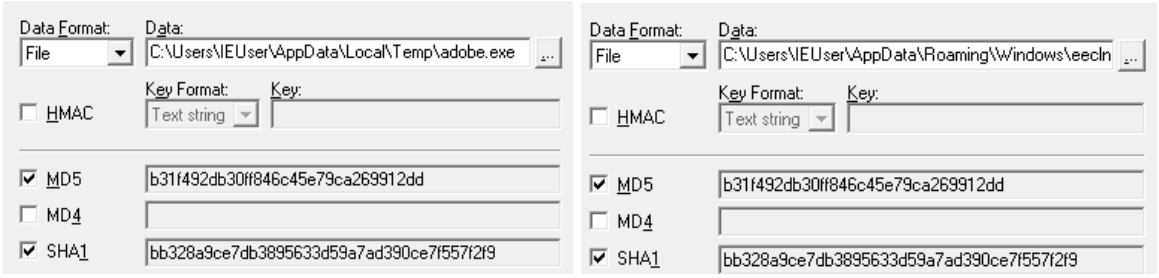

Fig 24: Same hashes between adobe.exe and eeclnt.exe

Comparing the other files between the 2 folders, both `MSVCR110.dll` and `MSVCR110.dat` created by the malware sample have the same hashes (Fig 25 and 26).

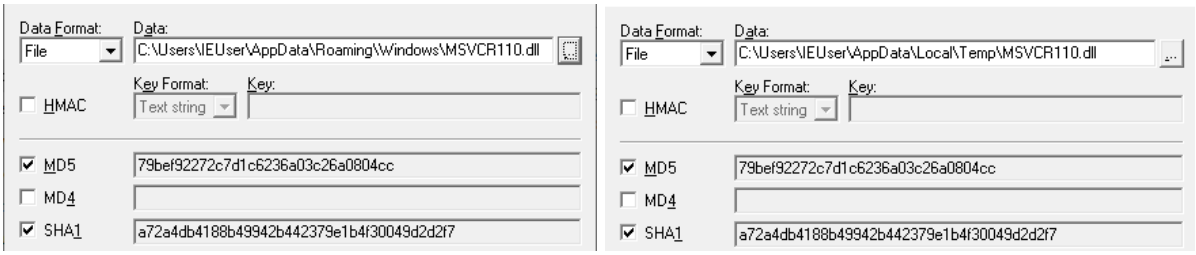

Fig 25: Created msvcr110.dll in different folders with the same hash

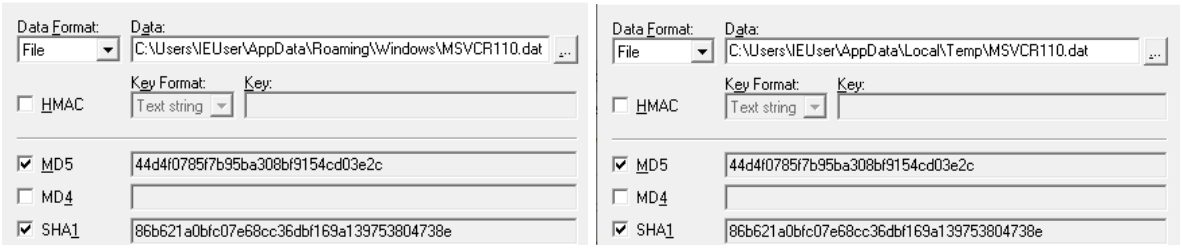

Fig 26: Created msvcr110.dat in different folders with the same hash

Moreover, looking at `MSVCR110.dll` of those found in the System folder (Fig 27), the hashes are found to be different from the ones created by the malware (Fig 25).

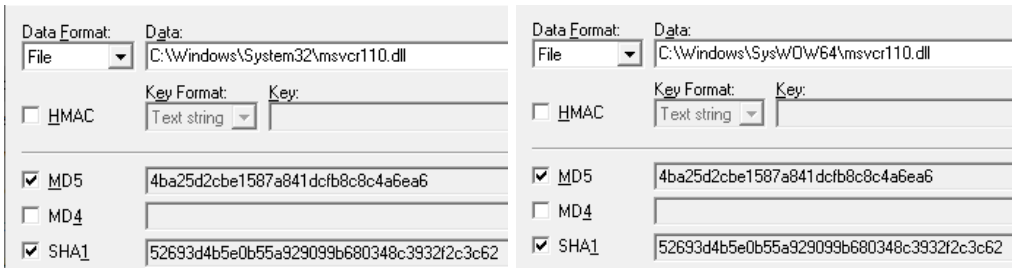

Fig 27: Hash of msvcr110.dll found in System folder

IDA:

After running the malware sample, we run IDA on the files created by the malware sample in an attempt to better understand the malware actions. It is observed that `adobe.exe` queries the registry for its values, likely to create, delete or edit registry key values (Fig 28).

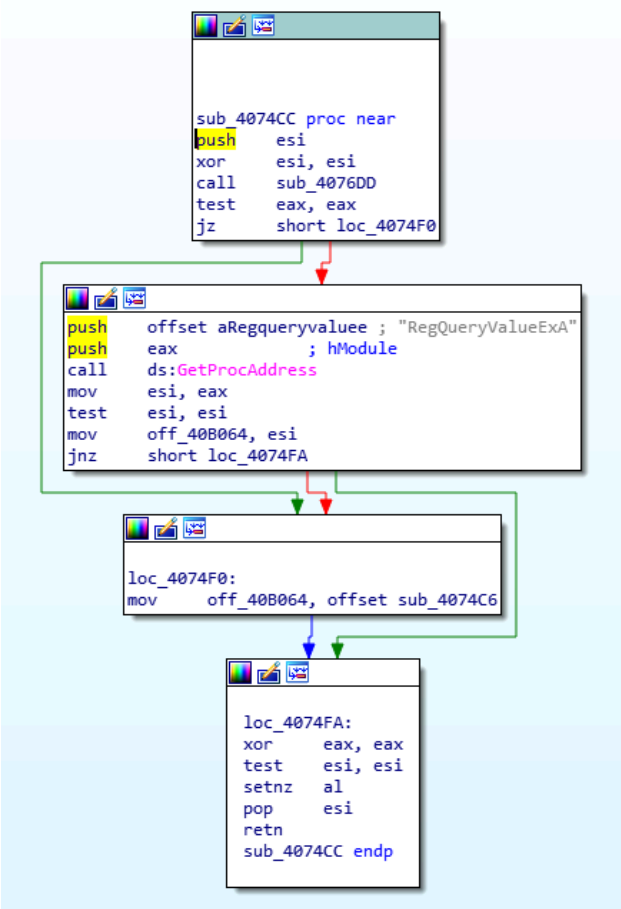

Fig 28: adobe.exe querying registry keys

It also checks if the current user is an admin user, and if so will likely use the admin privileges to execute its malicious commands (Fig 29).

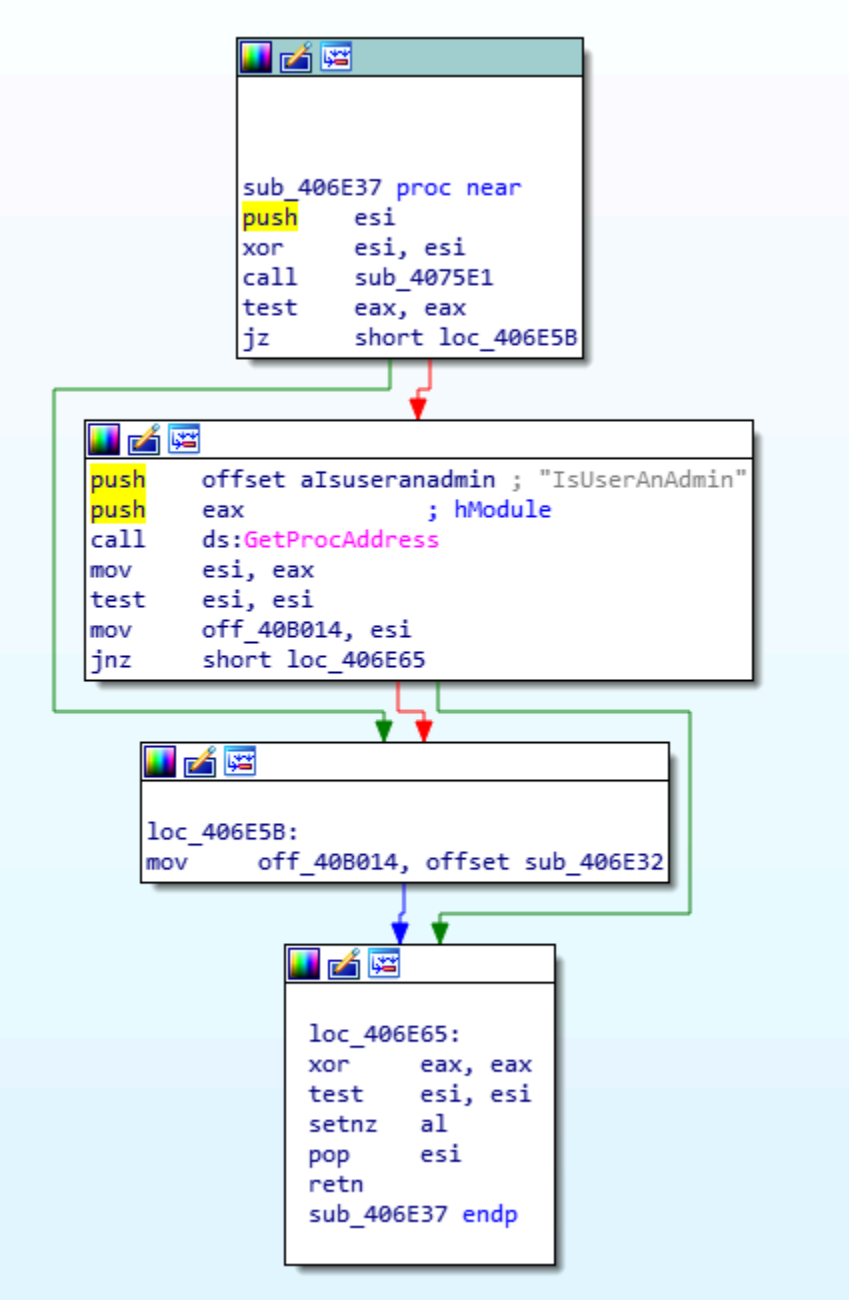

Fig 29: adobe.exe checking if the current user is an admin

Moreover, the `adobe.exe` executable also checks to see if it is a Wow64 process. This could indicate that the malware has the ability to run on different architecture, both on a 32 and 64 bit machine (Fig 30).

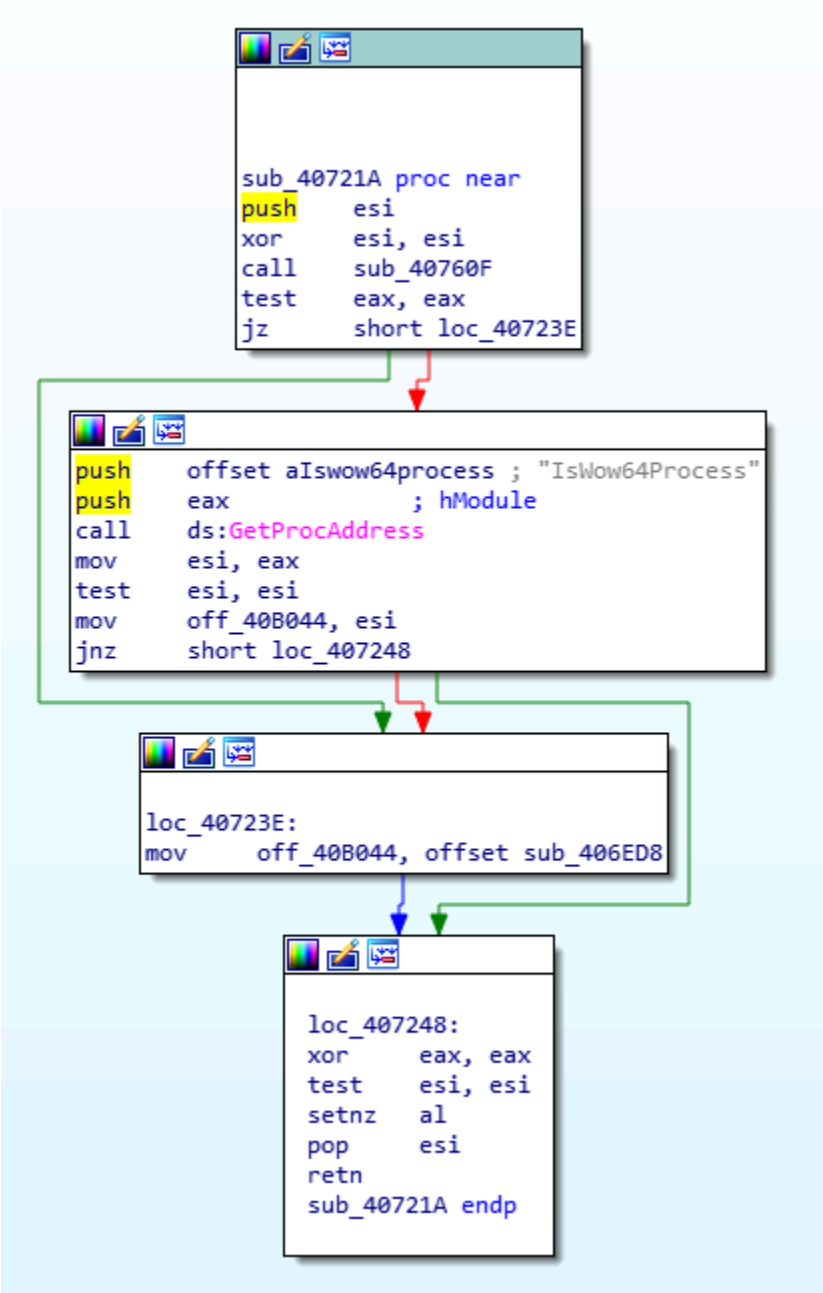

Fig 30: adobe.exe checking if it is a Wow64Process

ApateDNS:

When the malware sample first executes, ApateDNS captures that the malware sample tries to request for `news.singmicrosoft.ga` (Fig 31).

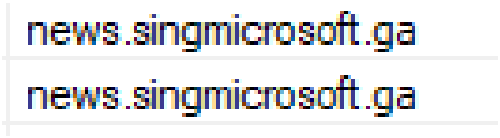

Fig 31: ApateDNS capturing what the malware attempts to connect to

These hosts might no longer be available as `Whois.com` is unable to find this domain (Fig 32).

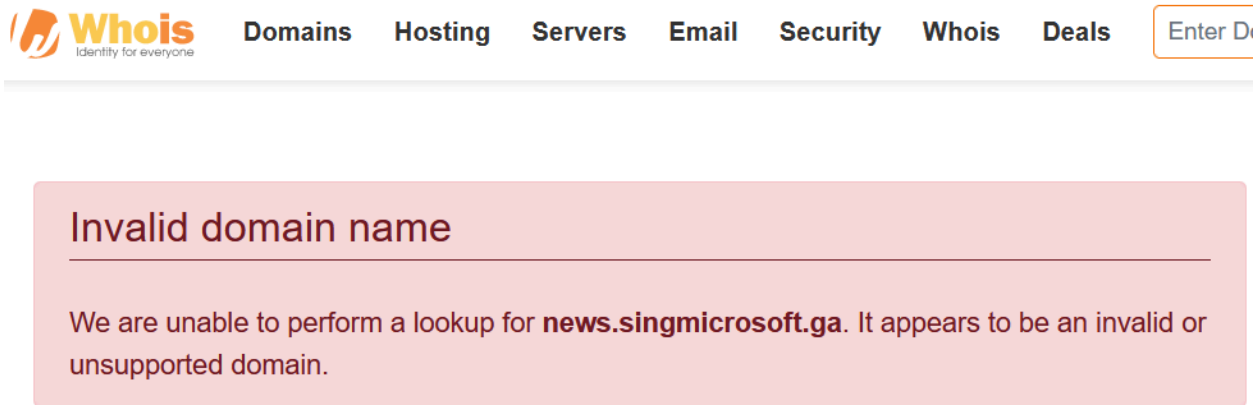

Fig 32: Whois.com unable to find the domain

Our assumption would be that this domain was used as the c2 server by the malicious actors.

ProcMon:

Using ProcMon we are able to confirm the actions of the malware sample observed previously when we attempted to run the malware sample. Firstly, we are able to see the creations of files by the original malware sample (Fig 33).

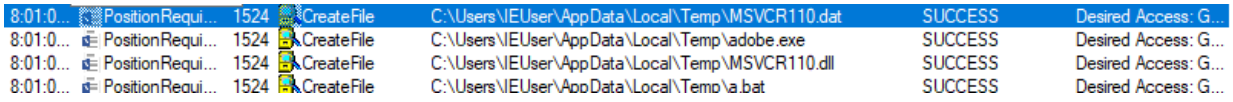

Fig 33: Malware sample creating suspicious files

The malware sample also then creates a copy of the actual word document to drop in the original folder as previously observed (Fig 34).

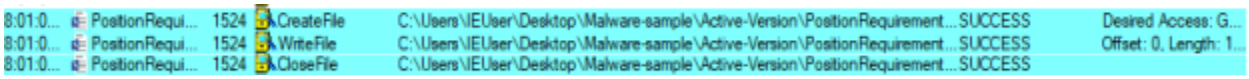

Fig 34: Malware sample creating legitimate word document

The malware sample then starts the process `adobe.exe` (Fig 35).

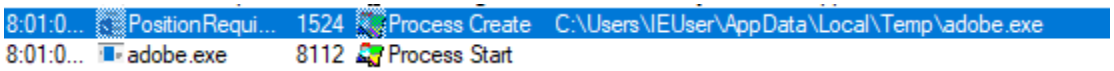

Fig 35: Malware sample starts adobe.exe process

We can also observe that `adobe.exe` then loads the malicious `MSVCR110.dll` (Fig 36).

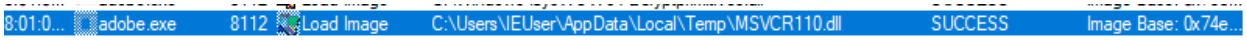

Fig 36: adobe.exe loads the malicious MSVCR110.dll

`adobe.exe` also starts the similar `eecInt.exe` executable (Fig 37).

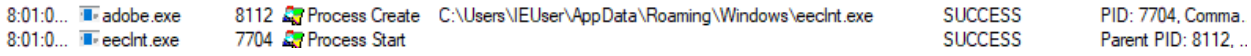

Fig 37: adobe.exe starting eecInt.exe

The new `eecInt.exe` then loads the malicious `MSVCR110.dll` and `MSVCR110.dat` similarly to `adobe.exe` (Fig 38).

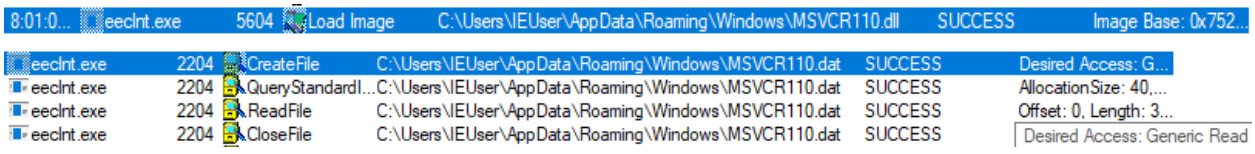

Fig 38: eecInt.exe loads the malicious MSVCR110.dll and MSVCR110.dat

The malware sample calls `cmd.exe` to allow the malware to execute commands (Fig 39).

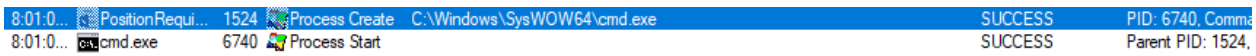

Fig 39: Malware sample starting the terminal process

ProcMon also shows us what `cmd.exe` accessed and can potentially execute (Fig 40).

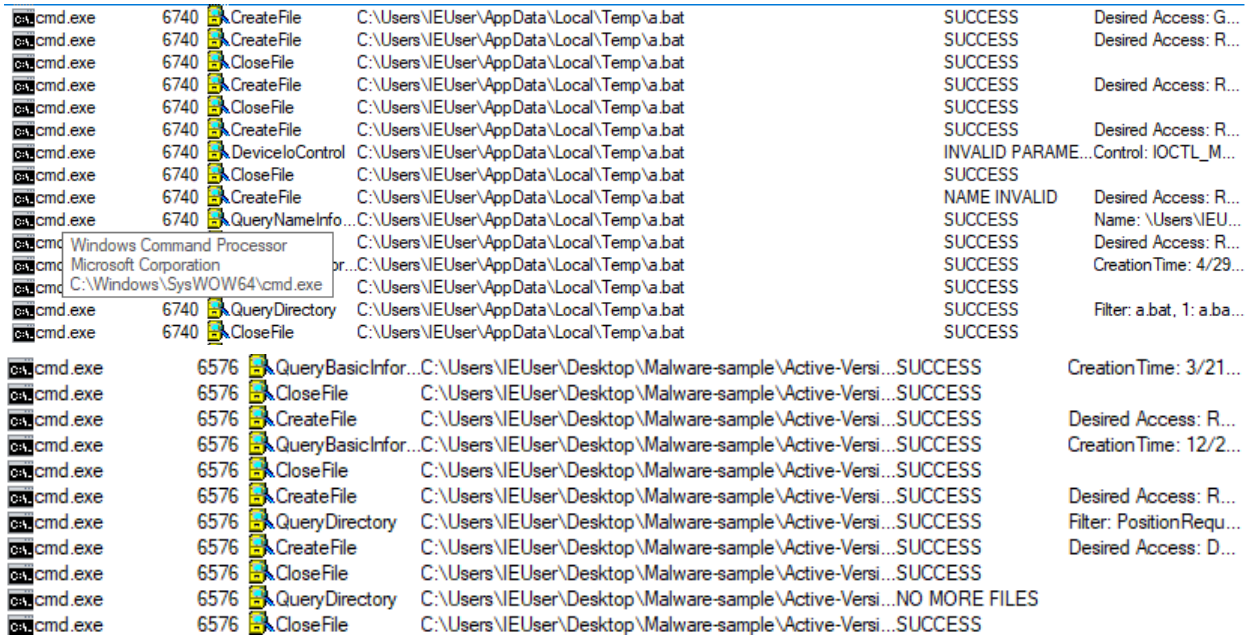

Fig 40: ProcMon showing which files from the malware sample cmd.exe executed

Regshot:

We used regshot to compare the changes in registry keys after running the malware. A key indicator which the malware sample has been executed would be to check the registry keys added (Fig 41).

```
----------------------------------
Keys added: 11
----------------------------------
HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\TermReason\10164
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\OneDrive\Installer\BITS\NucleusUpdateRingConfigJSON
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Applets\Wordpad
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Applets\Wordpad\Options
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Applets\Wordpad\Recent File List
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Applets\Wordpad\Settings
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\SessionInfo\1\ApplicationViewManagement\W32:
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\SessionInfo\1\ApplicationViewManagement\W32:
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\SessionInfo\1\ApplicationViewManagement\W32:
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\SessionInfo\1\ApplicationViewManagement\W32:
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\SessionInfo\1\ApplicationViewManagement\W32:
```

Fig 41: Snapshot of registry keys added

A registry key to focus which indicates the malware is present is the registry key below (Fig 42). This implies that `eeclnt.exe` will run whenever the user logs on with the command argument "260", similar to Fig 21 where the WanServer was started with a similar command.

```
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000
\Software\Microsoft\Windows\CurrentVersion\Run\eeclnt: "C:\Users\IEUser\AppData\Roaming\Windows\eeclnt.exe "260""
```

Fig 42: New registry key relating to eeclnt.exe

Another registry key added which shows the presence of the malware is this new registry key (Fig 43).

```
HKU\S-1-5-21-3461203602-4096304019-2269080069-1000\Software\Microsoft\Windows NT\CurrentVersion\
AppCompatFlags\Compatibility Assistant\Store\C:\Users\IEUser\Desktop\Malware-sample\Active-Version\
PositionRequirement-SeniorCivilEngineer.docx-53c1fd5ac99b5690b278ffcc5a49a598.exe:
53 41 43 50 01 00 00 00 00 00 00 00 07 00 00 00 28 00 00 00 00 74 03 00 2B E0 03 00 01 00 00 00 00 00 00 00 00
00 00 0A 71 22 00 00 67 07 7C BA C5 4C D4 01 00 00 00 00 00 00 00 00 02 00 00 00 28 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 D4 09 00 00 00 00 00 00 01 00 00 00 01 00 00 00
```

Fig 43: New registry key for the original malware sample

These new registry keys, along with the modifying of data of other existing registry keys is part of the techniques used for persistence.

Ollydbg:

Using Ollydbg, we can confirm the files that were created from running the malware from the previous analysis parts (Fig 44). Here are the files created when the malware was launched:

Fig 44: Files created seen in Ollydbg

From Fig 44, we see that “MSVCR110.dat”, “adobe.exe”, “MSVCR110.dll” and “a.bat” were created. As the malware continues running, it opens “adobe.exe” using the shell then the malware proceeds to create another file which seems to be a docx file with the same name as the malware and also opens it (Fig 45).

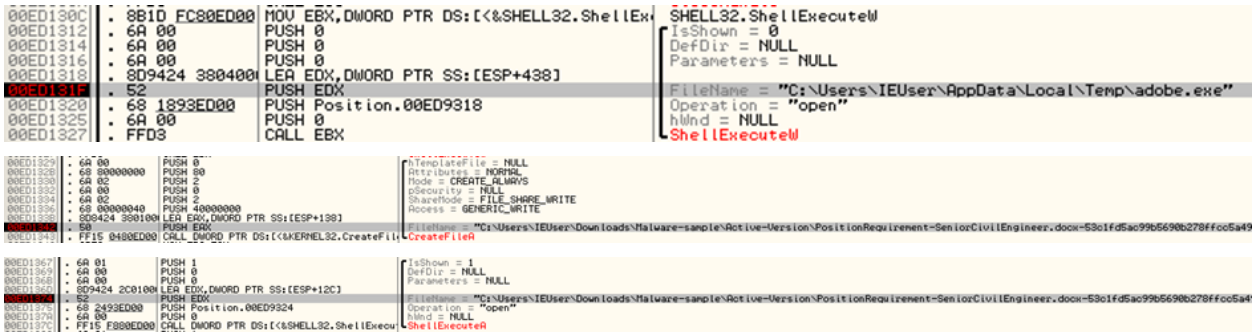

Fig 45: adobe.exe running which creates a docx file and also runs it

As previously found, there also exists the usage of an IsDebuggerPresent function that is used in the malware (Fig 15). This function can be found in the kernel32.dll file and checks whether the program is currently being debugged by a debugger like Ollydbg. As expected, the function returns true in the debugger and subsequently terminates the program (Fig 46).

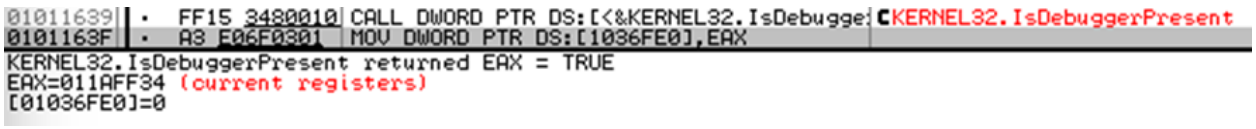

Fig 46: IsDebuggerPresent found to be used by the malware

Conclusion

In conclusion, this PlugX malware sample responsible for the SingHealth cybersecurity incident is a trojan that disguises itself as a non-suspicious word document but is actually an executable that executes various other activities in the background. We conducted a comprehensive analysis ranging from static analysis as well as dynamic analysis, conducting more static analysis on the newly created files by the malware as well.

Our findings from our static analysis shows that the malware is acting as a word document and aims to create and write new files to the system's temporary folder, subsequently executing them. Moreover, our dynamic analysis was consistent and supported the static analysis conducted previously. The malware aims to, through the use of dll sideloading, use the seemingly modified `MSVCR110.dll` to execute malicious code. Moreover, the malware might try to connect back with the c2 domain at `news.singmicrosoft.ga`. The malware will also maintain persistence via changing registry keys, adding multiple copies of itself and also be evasive by terminating itself in the presence of a debugger.

Annex A: Tools Used

1. Strings
2. HashCalc
3. PEView
4. Resource Hacker
5. IDA Free
6. ApateDNS
7. ProcMon
8. Regshot
9. Ollydbg