



**NUS**  
National University  
of Singapore

## **CS4238 Report**

### **Project 2 Team 4**

<b>Team Member</b>	<b>Matriculation No.</b>	<b>Email</b>
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

<b>Static Analysis.....</b>	<b>3</b>
VirusTotal:.....	3
PEView:.....	3
Strings:.....	5
Resource Hacker:.....	7
IDA:.....	8
<b>Dynamic Analysis.....</b>	<b>10</b>
IDA:.....	14
ApateDNS:.....	17
ProcMon:.....	18
Regshot:.....	20
Ollydbg:.....	21
<b>Conclusion.....</b>	<b>23</b>
<b>Annex A: Tools Used.....</b>	<b>24</b>

# Static Analysis

## VirusTotal:

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

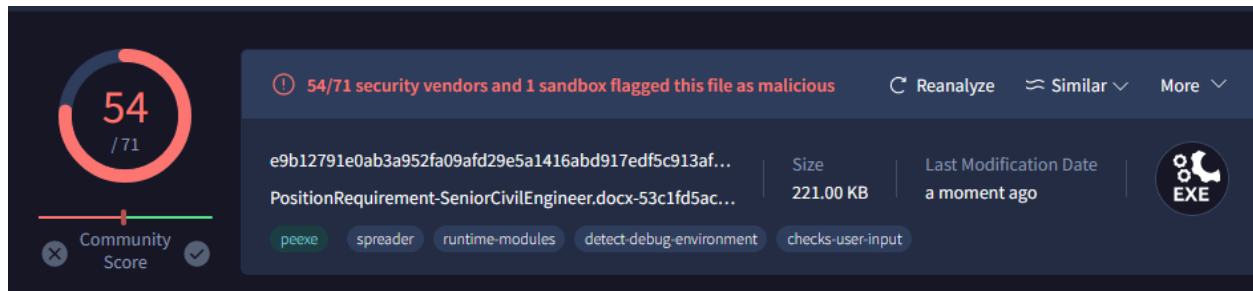


Fig 1: VirusTotal score for the malware sample

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.

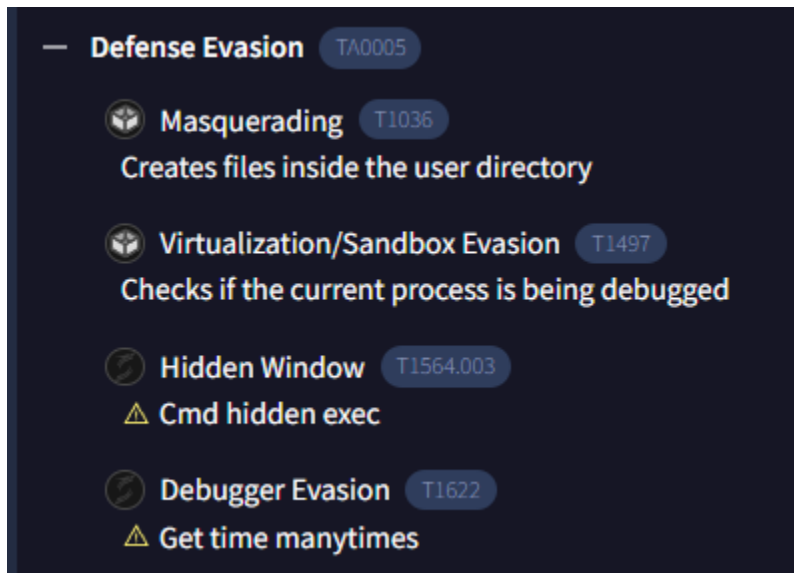


Fig 2: Malware capabilities by VirusTotal

## 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).

pFile	Data	Description	Value
00006A00	00009794	HintName RVA	0104 ExitProcess
00006A04	000097A2	HintName RVA	0078 CreateFileA
00006A08	000097B0	HintName RVA	0480 WriteFile
00006A0C	000097BC	HintName RVA	007F CreateFileW
00006A10	000097CA	HintName RVA	025B GetTempPathW
00006A14	000097DA	HintName RVA	01F4 GetModuleFileNameA
00006A18	000097F0	HintName RVA	0043 CloseHandle
00006A1C	0000980C	HintName RVA	019F GetCommandLineA
00006A20	0000980E	HintName RVA	0239 GetStartupInfoA
00006A24	00009800	HintName RVA	0420 TerminateProcess
00006A28	00009804	HintName RVA	01A0 GetCurrentProcess
00006A2C	000098A8	HintName RVA	043C UnhandledExceptionFilter
00006A30	000098C4	HintName RVA	0415 SetUnhandledExceptionFilter
00006A34	000098E2	HintName RVA	02D1 IsDebuggerPresent
00006A38	000098F6	HintName RVA	01F3 GetModuleHandleW
00006A3C	0000990A	HintName RVA	0421 Sleep
00006A40	00009912	HintName RVA	0220 GetProcAddress
00006A44	00009924	HintName RVA	0230 GetStdHandle
00006A48	00009934	HintName RVA	01AA FreeEnvironmentStringsA
00006A4C	0000994E	HintName RVA	010F GetEnvironmentStrings
00006A50	00009966	HintName RVA	014B FreeEnvironmentStringsW
00006A54	00009980	HintName RVA	047A WideCharToMultiByte
00006A58	00009996	HintName RVA	01E6 GetLastErrMsg
00006A5C	000099A6	HintName RVA	01C1 GetEnvironmentStringsW
00006A60	000099C0	HintName RVA	03E8 SetHandleCount
00006A64	000099D2	HintName RVA	01D7 GetFileType
00006A68	000099E0	HintName RVA	009E DeleteCriticalSection
00006A6C	000099F0	HintName RVA	0434 TlsGetValue
00006A70	00009A06	HintName RVA	0432 TlsAlloc
00006A74	00009A12	HintName RVA	0435 TlsSetValue
00006A78	00009A20	HintName RVA	0433 TlsFree
00006A7C	00009A2A	HintName RVA	02C0 InterlockedIncrement
00006A80	00009A42	HintName RVA	03EC SetLastError
00006A84	00009A52	HintName RVA	01AD GetCurrentThreadId
00006A88	00009A68	HintName RVA	02BC InterlockedDecrement
00006A8C	00009A80	HintName RVA	029F HeapCreate
00006A90	00009A8E	HintName RVA	0457 VirtualFree
00006A94	00009A9C	HintName RVA	02A1 HeapFree
00006A98	00009AAB	HintName RVA	0254 QueryPerformanceCounter
00006A9C	00009AC2	HintName RVA	0256 GetTickCount
00006AA0	00009AD2	HintName RVA	01AA GetCurrentProcessId
00006AA4	00009AE8	HintName RVA	024F GetSystemTimeAsFileTime
00006AA8	00009B02	HintName RVA	02EF LeaveCriticalSection
00006AAC	00009B1A	HintName RVA	00D9 EnterCriticalSection
00006AAE	00009B32	HintName RVA	02F1 LoadImageA
00006AB4	00009B42	HintName RVA	0255 InitializeCriticalSectionAndSpinCount
00006AB8	00009B5A	HintName RVA	010B GetCPInfo
00006ABC	00009B76	HintName RVA	0102 GetACP
00006AC0	00009B88	HintName RVA	0213 GetDeviceCP
00006AC4	00009B9C	HintName RVA	020B IsValidCodePage
00006AC8	00009B9E	HintName RVA	0290 HeapAlloc
00006ACC	00009BAA	HintName RVA	0454 VirtualAlloc
00006AD0	00009BBA	HintName RVA	02A4 HeapReAlloc
00006AD4	00009BC8	HintName RVA	0362 ReUnwind
00006AD8	00009BD4	HintName RVA	02A6 HeapSize
00006ADC	00009BED	HintName RVA	01E8 GetLocalTimeA
00006AE0	00009BF2	HintName RVA	02E1 LCMagStringA
00006AE4	00009C02	HintName RVA	031A MultiByteToWideChar
00006AE8	00009C18	HintName RVA	02E3 LCMagStringW
00006AEC	00009C28	HintName RVA	02D0 GetStringTypeA
00006AF0	00009C3A	HintName RVA	0240 GetStringTypeW
00006AF4	00009C90	End of Imports	KERNEL32.dll
00006AF8	00009C30	HintName RVA	0114 ShellExecuteA
00006AFC	00009C40	HintName RVA	0118 ShellExecuteW
00006B00	00009C90	End of Imports	SHELL32.dll
00006B04	00009C0C	HintName RVA	0306 InspireW
00006B08	00009C18	HintName RVA	0307 InspireA
00006B0C	00009C90	End of Imports	USER32.dll

Fig 4: Imported functions and libraries by the malware sample

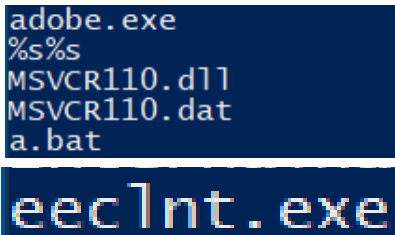
**ShellExecuteA** and **ShellExecuteW** are functions used to execute a specified file or program. **CreateFileA** and **CreateFileW** can be used to create or open existing files. **WriteFile** writes data to a file. These functions can allow the malware to write and alter data to then execute them.

**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:

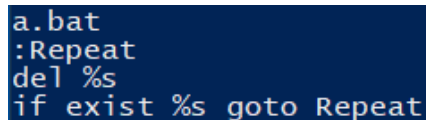
Running `strings` on the malware returns multiple strings which could be of interest. This includes: `adobe.exe`, `MSVCR100.dll`, `MSVCR110.dat`, `eeclnt.exe` and `a.bat` batch script file (Fig 5).



```
adobe.exe
%s%s
MSVCR110.dll
MSVCR110.dat
a.bat
eeclnt.exe
```

Fig 5: Filenames found in strings

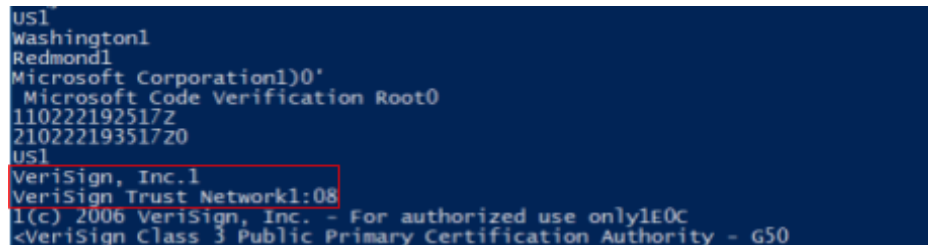
Moreover, additional strings are found after the `a.bat` file (Fig 6). This could potentially contain the code for the `a.bat`. `%s` could indicate a string variable and `del` command deletes a file in the current folder with the name in the `%s` variable. Hence, `%s` could potentially be the filename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



```
a.bat
:Repeat
del %s
if exist %s goto Repeat
```

Fig 6: `a.bat` batch script

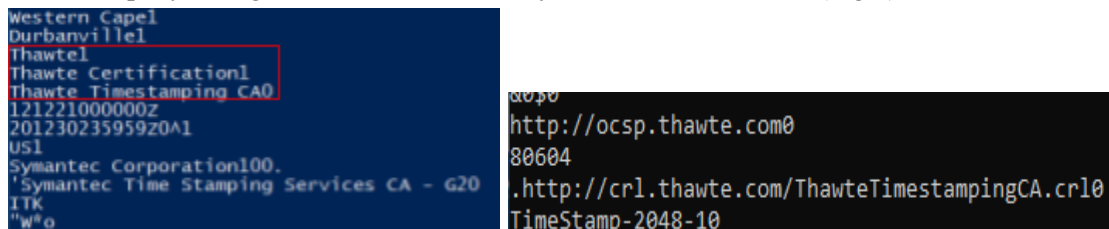
Certificate signing strings have also been discovered. Verisign Inc. is a company that operates a diverse array of network infrastructure, including two of the Internet's thirteen root name servers (Fig 7).



```
US1
Washington1
Redmond1
Microsoft Corporation1)0'
Microsoft Code Verification Root0
110222192517Z
210222193517Z0
US1
VeriSign, Inc.1
VeriSign Trust Network1:08
1(c) 2006 VeriSign, Inc. - For authorized use only1E0C
<VeriSign Class 3 Public Primary Certification Authority - G50
```

Fig 7: VeriSign string

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



```
Western Capel
Durbanville1
Thawte1
Thawte Certification1
Thawte Timestamping CA0
121221000000Z
201230235959Z0A1
US1
Symantec Corporation100.
Symantec Time Stamping Services CA - G20
ITK
"WO
http://ocsp.thawte.com0
80604
http://crl.thawte.com/ThawteTimestampingCA.crl0
TimeStamp-2048-10
```

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.

```
Symantec Corporation100.
'Symantec Time Stamping Services CA - G20
121018000000Z
201229235959Z0b1
US1
Symantec Corporation1402
+Symantec Time Stamping Services Signer - G40
<Su
CK"
2oNW
a;EQ
f=G
g0e0*
http://ts-ocsp.ws.symantec.com07
+http://ts-aia.ws.symantec.com/tss-ca-g2.cer0<
50301
+http://ts-crl.ws.symantec.com/tss-ca-g2.crl0(
TimeStamp-2048-20
```

Fig 9: Symantec Time Stamping Service String

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

```
StringFileInfo
040904e4
CompanyName
ESET
FileDescription
ESET Elevated Client
FileVersion
8.0.319.0
InternalName
eeclnt.exe
LegalCopyright
Copyright (c) ESET, spol. s r.o. 1992-2015. All rights reserved.
LegalTrademarks
NOD, NOD32, AMON, ESET are registered trademarks of ESET.
OriginalFilename
eeclnt.exe
ProductName
ESET Smart Security
ProductVersion
8.0.319.0
VarFileInfo
```

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.

```
WTSGetActiveConsoleSessionId
shell32.dll
kernel32.dll
psapi.dll
ntdll.dll
advapi32.dll
```

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.

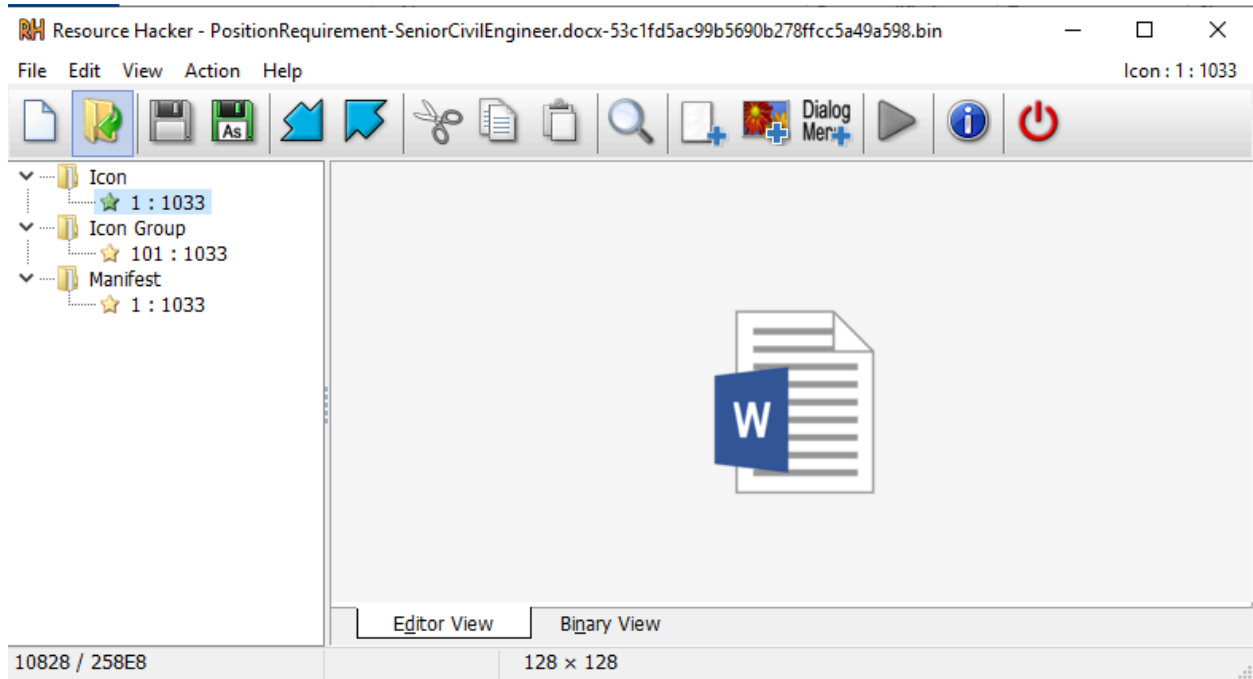


Fig 12: Word Document icon resource found in the malware sample

## 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.

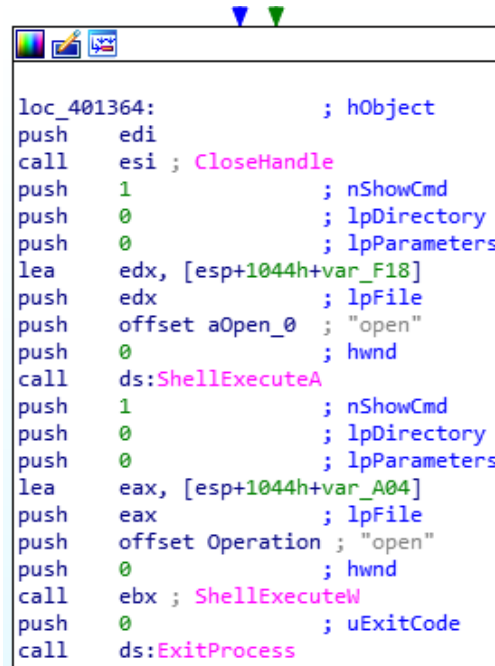
```
mov     [esp+1074h+var_A04], dx
call    sub_405730
add     esp, 3Ch
lea     ecx, [esp+1038h+Buffer]
push    ecx                ; lpBuffer
push    104h               ; nBufferLength
call    ds:GetTempPathW
mov     esi, ds:wsprintfW
push    offset aAdobeExe ; "adobe.exe"
lea     edx, [esp+103Ch+Buffer]
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
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`



```

loc_401364:                ; hObject
push     edi
call     esi ; CloseHandle
push     1                  ; nShowCmd
push     0                  ; lpDirectory
push     0                  ; lpParameters
lea      edx, [esp+1044h+var_F18]
push     edx                ; lpFile
push     offset aOpen_0     ; "open"
push     0                  ; hWnd
call     ds:ShellExecuteA
push     1                  ; nShowCmd
push     0                  ; lpDirectory
push     0                  ; lpParameters
lea      eax, [esp+1044h+var_A04]
push     eax                ; lpFile
push     offset Operation   ; "open"
push     0                  ; hWnd
call     ebx ; ShellExecuteW
push     0                  ; uExitCode
call     ds:ExitProcess

```

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.

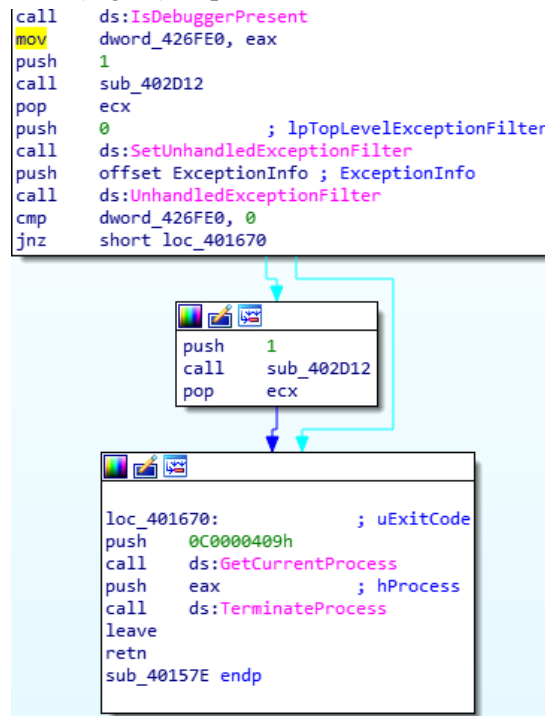


Fig 15: Malware sample checking if it is in a debugging environment

# 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).

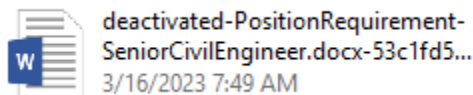


Fig 16: Exe file changes thumbnail icon to a word doc

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.

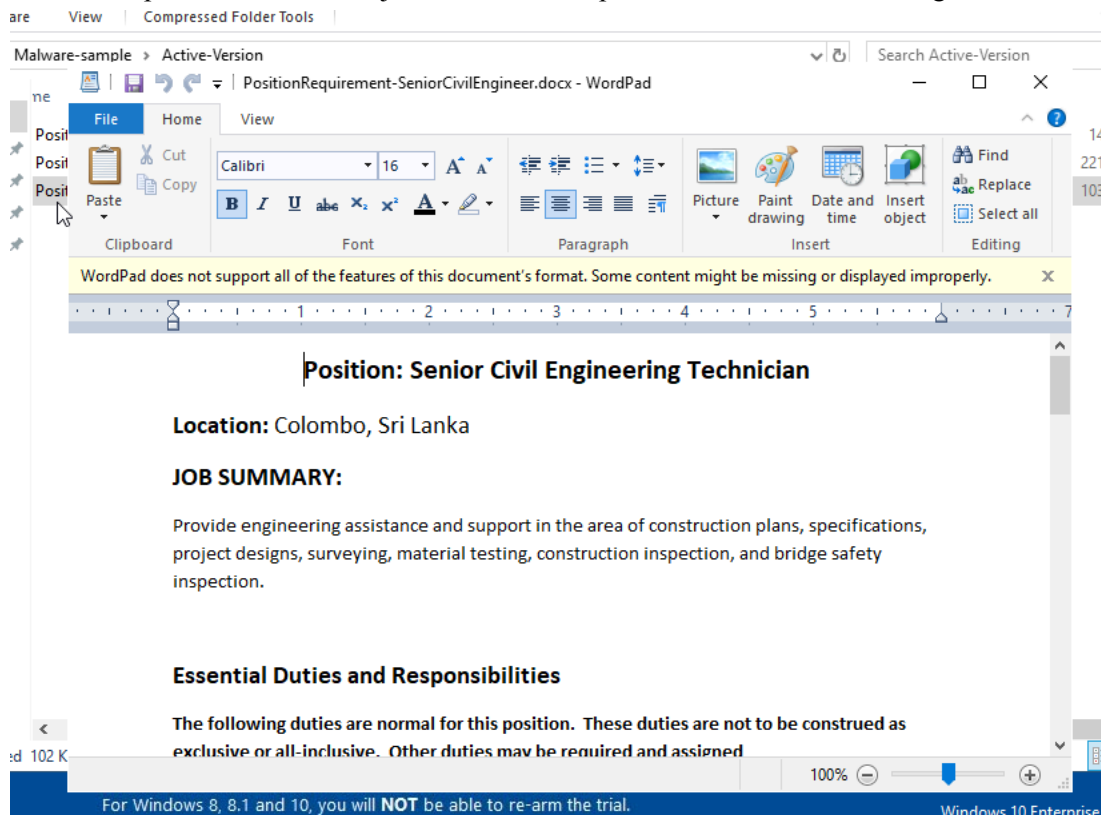


Fig 17: Word Doc that opens after running the malware

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).

	Date modified	Type	Size
nRequirement-SeniorCivilEngineer.docx	4/27/2024 8:22 AM	Office Open XML ...	14 KB
nRequirement-SeniorCivilEngineer.docx-53c1fd5ac99b5690b278ffcc5a49a598.bin	12/28/2020 6:12 PM	BIN File	221 KB

Fig 18: File size decrease

From the previous static analysis, we noted that the malware created `adobe.exe`, `MSVCR110.dll`, `MSVCR110.dat`, `eecInt.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).

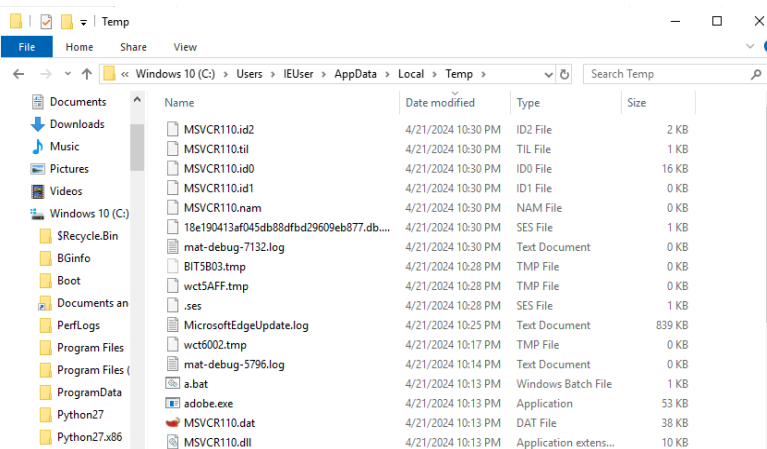


Fig 19: Temp folder containing the files which the malware created

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.

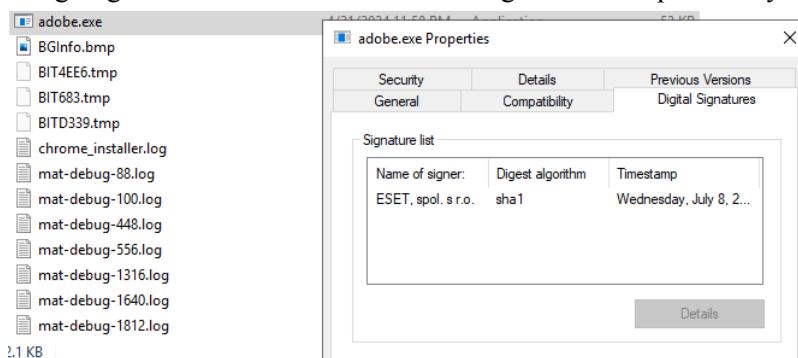


Fig 20: Signed adobe.exe by ESET

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.

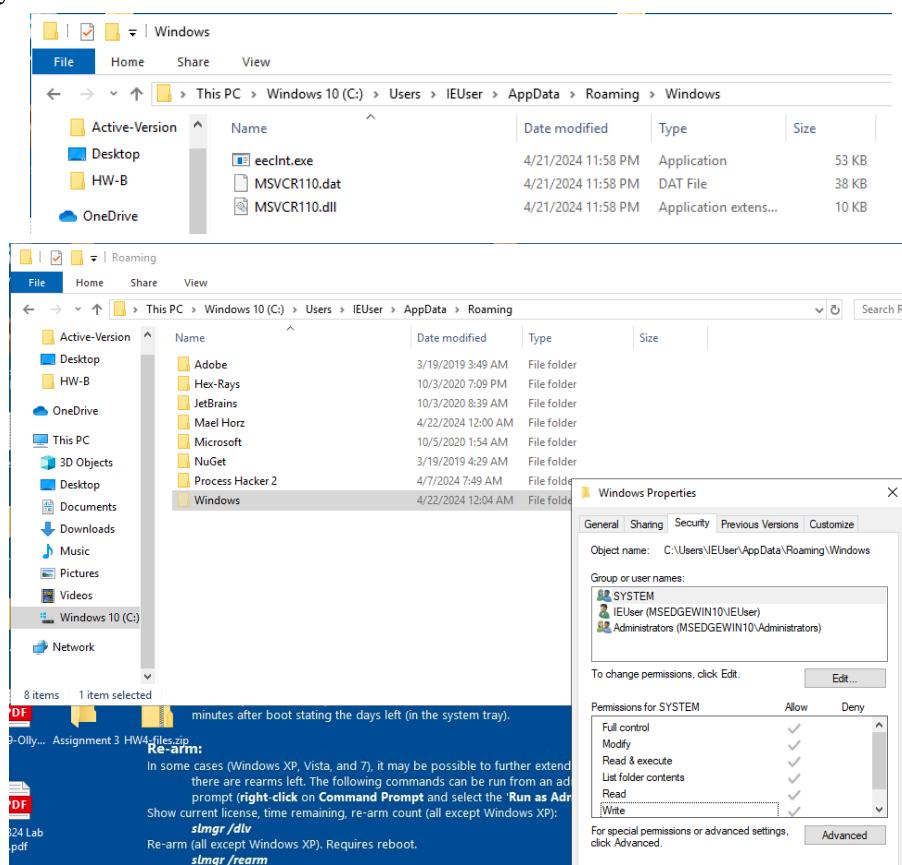


Fig 22: Files created by malware in AppData\Roaming\Windows

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

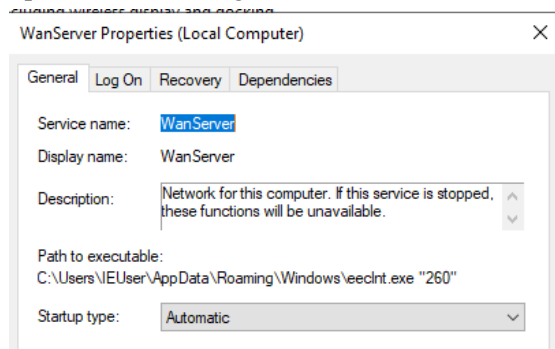


Fig 23: WanServer created

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**.

Tool	Data Format	Data	MD5	SHA1
Left	File	C:\Users\IEUser\AppData\Local\Temp\adobe.exe	b31f492db30ff846c45e79ca269912dd	bb328a9ce7db3895633d59a7ad390ce7f557f2f9
Right	File	C:\Users\IEUser\AppData\Roaming\Windows\eeclnt.exe	b31f492db30ff846c45e79ca269912dd	bb328a9ce7db3895633d59a7ad390ce7f557f2f9

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).

Tool	Data Format	Data	MD5	SHA1
Left	File	C:\Users\IEUser\AppData\Roaming\Windows\MSVCR110.dll	79bef92272c7d1c6236a03c26a0804cc	a72a4db4188b49942b442379e1b4f30049d2d2f7
Right	File	C:\Users\IEUser\AppData\Local\Temp\MSVCR110.dll	79bef92272c7d1c6236a03c26a0804cc	a72a4db4188b49942b442379e1b4f30049d2d2f7

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

Tool	Data Format	Data	MD5	SHA1
Left	File	C:\Users\IEUser\AppData\Roaming\Windows\MSVCR110.dat	44d4f0785f7b95ba308bf9154cd03e2c	86b621a0bfc07e68cc36dbf169a139753804738e
Right	File	C:\Users\IEUser\AppData\Local\Temp\MSVCR110.dat	44d4f0785f7b95ba308bf9154cd03e2c	86b621a0bfc07e68cc36dbf169a139753804738e

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).

Tool	Data Format	Data	MD5	SHA1
Left	File	C:\Windows\System32\msvcr110.dll	4ba25d2cbe1587a841dcfb8c8c4a6ea6	52693d4b5e0b55a929099b680348c3932f2c3c62
Right	File	C:\Windows\System64\msvcr110.dll	4ba25d2cbe1587a841dcfb8c8c4a6ea6	52693d4b5e0b55a929099b680348c3932f2c3c62

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).

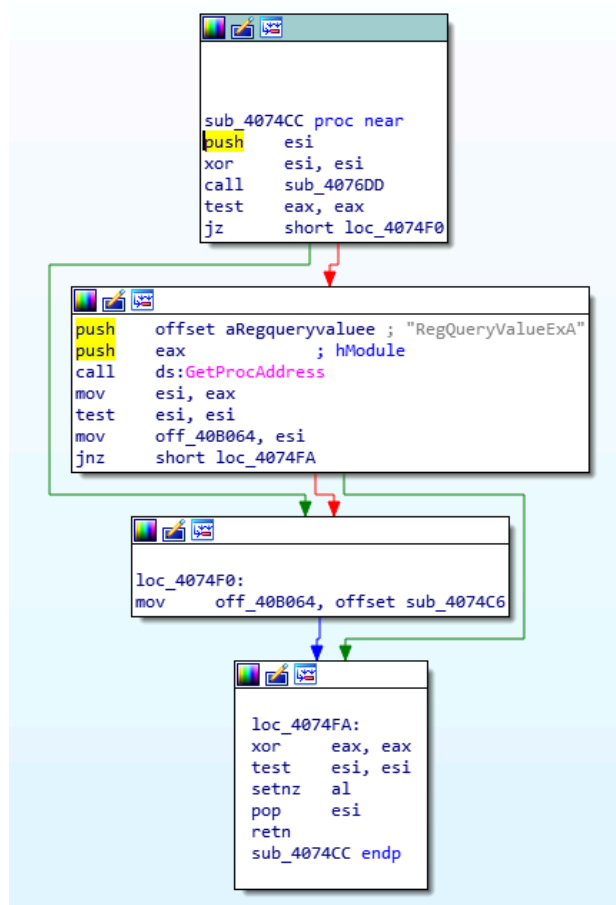


Fig 28: adobe.exe querying registry keys

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).

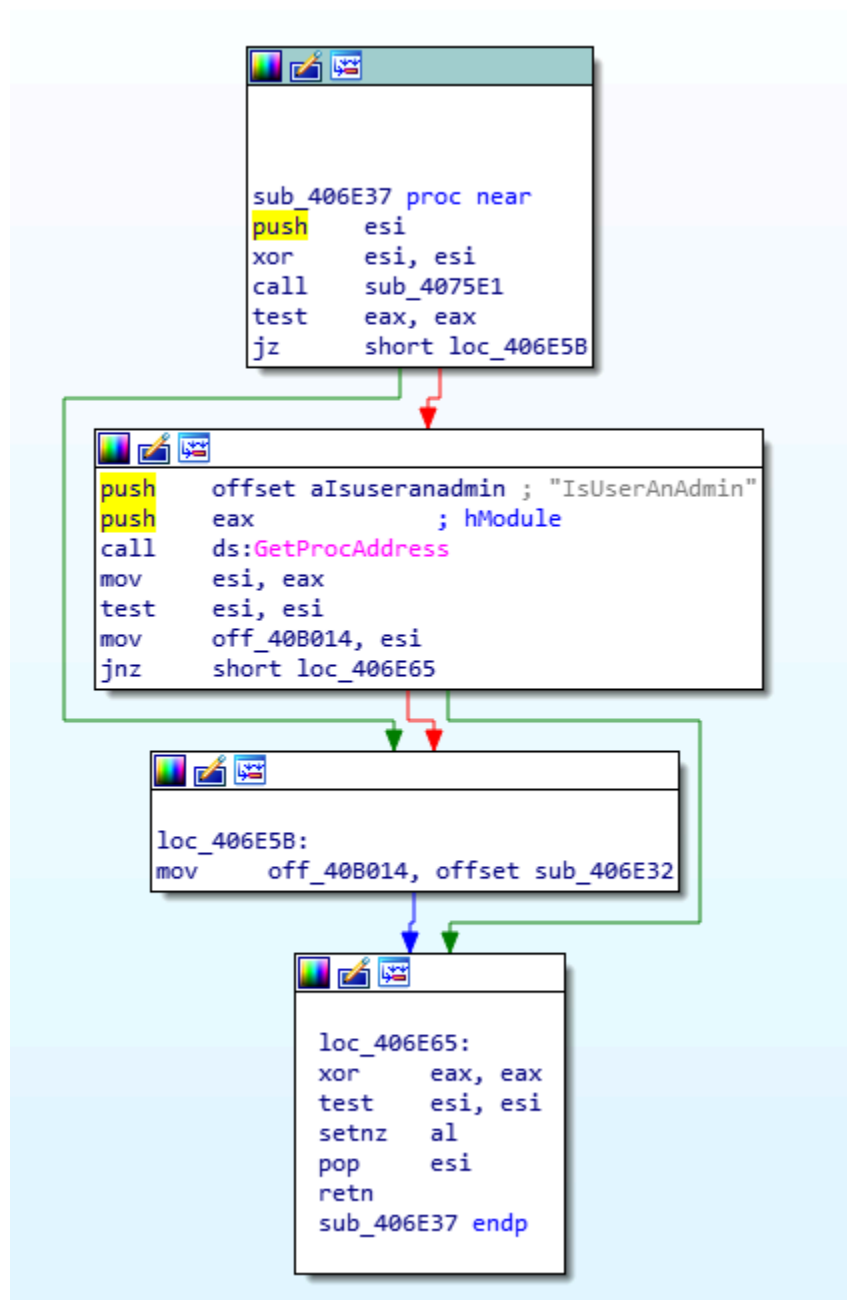


Fig 29: adobe.exe checking if the current user is an admin

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).

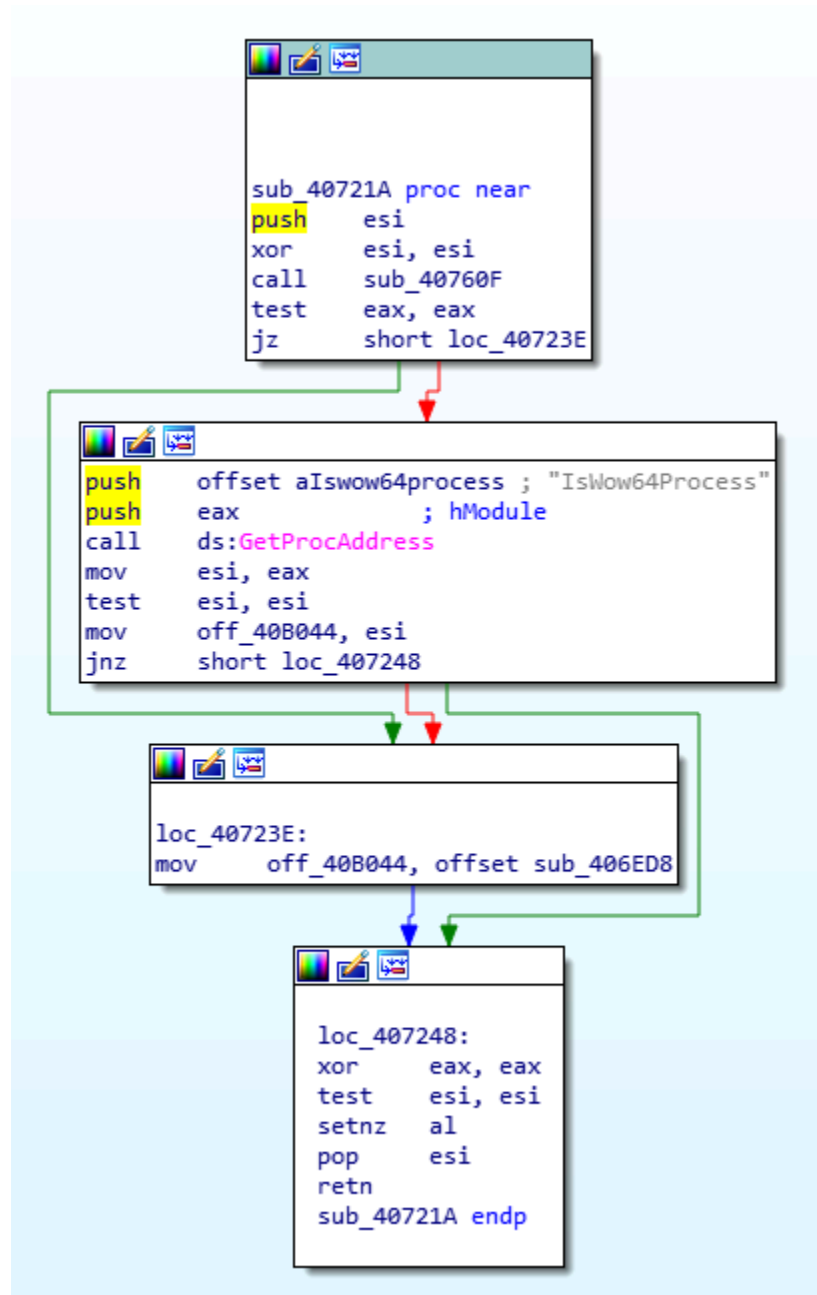


Fig 30: adobe.exe checking if it is a Wow64Process



## ApateDNS:

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

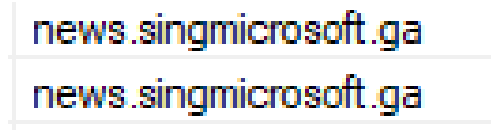


Fig 31: ApateDNS capturing what the malware attempts to connect to

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

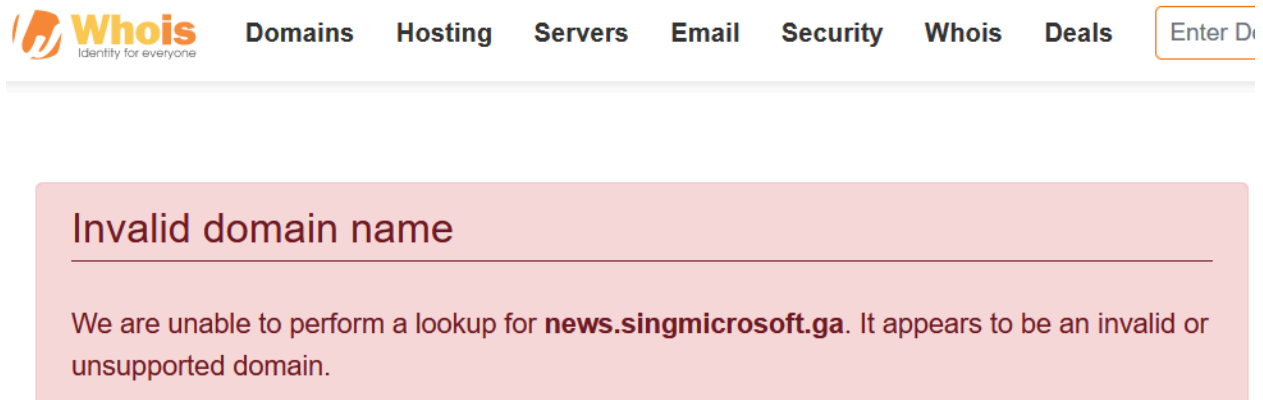
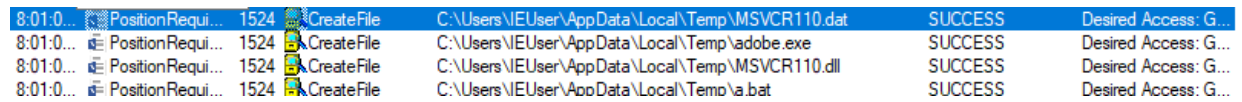


Fig 32: Whois.com unable to find the domain

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

## ProcMon:

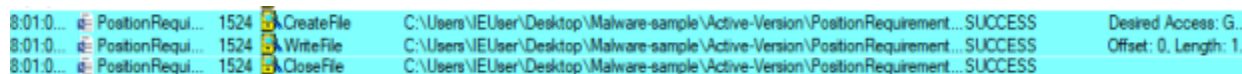
Using ProcMon we are able to confirm the actions of the malware sample observed previously when we attempted to run the malware sample. Firstly, we are able to see the creations of files by the original malware sample (Fig 33).



8:01:0...	PositionRequi...	1524	CreateFile	C:\Users\IEUser\AppData\Local\Temp\MSVCR110.dat	SUCCESS	Desired Access: G...
8:01:0...	PositionRequi...	1524	CreateFile	C:\Users\IEUser\AppData\Local\Temp\adobe.exe	SUCCESS	Desired Access: G...
8:01:0...	PositionRequi...	1524	CreateFile	C:\Users\IEUser\AppData\Local\Temp\MSVCR110.dll	SUCCESS	Desired Access: G...
8:01:0...	PositionRequi...	1524	CreateFile	C:\Users\IEUser\AppData\Local\Temp\adobe.bat	SUCCESS	Desired Access: G...

Fig 33: Malware sample creating suspicious files

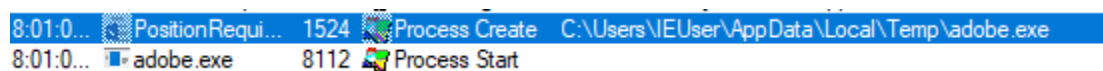
The malware sample also then creates a copy of the actual word document to drop in the original folder as previously observed (Fig 34).



8:01:0...	PositionRequi...	1524	CreateFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Version\PositionRequirement...	SUCCESS	Desired Access: G...
8:01:0...	PositionRequi...	1524	WriteFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Version\PositionRequirement...	SUCCESS	Offset: 0, Length: 1...
8:01:0...	PositionRequi...	1524	CloseFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Version\PositionRequirement...	SUCCESS	

Fig 34: Malware sample creating legitimate word document

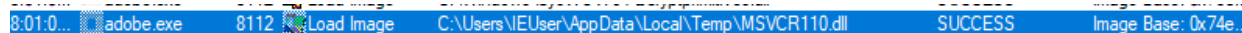
The malware sample then starts the process `adobe.exe` (Fig 35).



8:01:0...	PositionRequi...	1524	Process Create	C:\Users\IEUser\AppData\Local\Temp\adobe.exe		
8:01:0...	adobe.exe	8112	Process Start			

Fig 35: Malware sample starts adobe.exe process

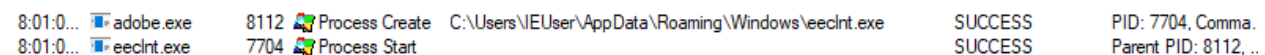
We can also observe that `adobe.exe` then loads the malicious `MSVCR110.dll` (Fig 36).



8:01:0...	adobe.exe	8112	Load Image	C:\Users\IEUser\AppData\Local\Temp\MSVCR110.dll	SUCCESS	Image Base: 0x74e...
-----------	-----------	------	------------	---	---------	----------------------

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

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



8:01:0...	adobe.exe	8112	Process Create	C:\Users\IEUser\AppData\Roaming\Windows\eeclnt.exe	SUCCESS	PID: 7704, Comm...
8:01:0...	eeclnt.exe	7704	Process Start		SUCCESS	Parent PID: 8112, ...

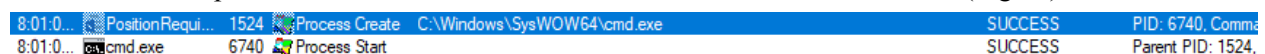
Fig 37: adobe.exe starting eeclnt.exe

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

8:01:0...	eeclnt.exe	5604	Load Image	C:\Users\IEUser\AppData\Roaming\Windows\MSVCR110.dll	SUCCESS	Image Base: 0x752...
eeclnt.exe	2204	CreateFile	C:\Users\IEUser\AppData\Roaming\Windows\MSVCR110.dat	SUCCESS	Desired Access: G...	
eeclnt.exe	2204	QueryStandard...	C:\Users\IEUser\AppData\Roaming\Windows\MSVCR110.dat	SUCCESS	AllocationSize: 40,...	
eeclnt.exe	2204	ReadFile	C:\Users\IEUser\AppData\Roaming\Windows\MSVCR110.dat	SUCCESS	Offset: 0, Length: 3...	
eeclnt.exe	2204	CloseFile	C:\Users\IEUser\AppData\Roaming\Windows\MSVCR110.dat	SUCCESS	Desired Access: Generic Read	

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

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



8:01:0...	PositionRequi...	1524	Process Create	C:\Windows\SysWOW64\cmd.exe	SUCCESS	PID: 6740, Comm...
8:01:0...	cmd.exe	6740	Process Start		SUCCESS	Parent PID: 1524,

Fig 39: Malware sample starting the terminal process

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

cmd.exe	6740	CreateFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	Desired Access: G...
cmd.exe	6740	CreateFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	Desired Access: R...
cmd.exe	6740	CloseFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	
cmd.exe	6740	CreateFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	Desired Access: R...
cmd.exe	6740	CloseFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	
cmd.exe	6740	CreateFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	Desired Access: R...
cmd.exe	6740	DeviceIoControl	C:\Users\IEUser\AppData\Local\Temp\*.bat	INVALID PARAMETER	Control: IOCTL_M...
cmd.exe	6740	CloseFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	
cmd.exe	6740	CreateFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	NAME INVALID	Desired Access: R...
cmd.exe	6740	QueryNameInfo	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	Name: \Users\IEU...
cmd.exe		Windows Command Processor	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	Desired Access: R...
cmd.exe		Microsoft Corporation	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	CreationTime: 4/29...
cmd.exe		C:\Windows\SysWOW64\cmd.exe	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	
cmd.exe	6740	QueryDirectory	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	Filter: *.bat, 1: *.ba...
cmd.exe	6740	CloseFile	C:\Users\IEUser\AppData\Local\Temp\*.bat	SUCCESS	
cmd.exe	6576	QueryBasicInfo	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	CreationTime: 3/21...
cmd.exe	6576	CloseFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	
cmd.exe	6576	CreateFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	Desired Access: R...
cmd.exe	6576	QueryBasicInfo	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	CreationTime: 12/2...
cmd.exe	6576	CloseFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	
cmd.exe	6576	CreateFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	Desired Access: R...
cmd.exe	6576	QueryDirectory	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	Filter: PositionRequ...
cmd.exe	6576	CreateFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	Desired Access: D...
cmd.exe	6576	CloseFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	
cmd.exe	6576	QueryDirectory	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	NO MORE FILES	
cmd.exe	6576	CloseFile	C:\Users\IEUser\Desktop\Malware-sample\Active-Versi...	SUCCESS	

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 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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:

00ED1144	6A 00	PUSH 0	hTemplateFile = NULL
00ED1146	68 80000000	PUSH 80	Attributes = NORMAL
00ED1148	6A 02	PUSH 2	Mode = CREATE_ALWAYS
00ED114D	6A 00	PUSH 0	pSecurity = NULL
00ED114F	6A 02	PUSH 2	ShareMode = FILE_SHARE_WRITE
00ED1151	68 00000040	PUSH 40000000	Access = GENERIC_WRITE
00ED1156	8B35 0C80ED00	MOV ESI,DWORD PTR DS:[<&KERNEL32.CreateFileW	KERNEL32.CreateFileW
00ED115C	8D8C24 540800	LEA ECX,DWORD PTR SS:[ESP+854]	FileName = "C:\Users\IEUser\AppData\Local\Temp\MSVCR110.dat"
00ED1163	51	PUSH ECX	CreateFileW
00ED1164	FFD6	CALL ESI	
00ED1166	6A 00	PUSH 0	hTemplateFile = NULL
00ED1168	68 80000000	PUSH 80	Attributes = NORMAL
00ED116D	6A 02	PUSH 2	Mode = CREATE_ALWAYS
00ED116F	6A 00	PUSH 0	pSecurity = NULL
00ED1171	6A 02	PUSH 2	ShareMode = FILE_SHARE_WRITE
00ED1173	68 00000040	PUSH 40000000	Access = GENERIC_WRITE
00ED1178	8D9424 440400	LEA EDX,DWORD PTR SS:[ESP+444]	FileName = "C:\Users\IEUser\AppData\Local\Temp\adobe.exe"
00ED117F	52	PUSH EDX	CreateFileW
00ED1180	8BF8	MOV EDI,EBX	
00ED1182	FFD6	CALL ESI	
00ED1184	6A 00	PUSH 0	hTemplateFile = NULL
00ED1186	68 80000000	PUSH 80	Attributes = NORMAL
00ED118B	6A 02	PUSH 2	Mode = CREATE_ALWAYS
00ED118D	6A 00	PUSH 0	pSecurity = NULL
00ED118F	6A 02	PUSH 2	ShareMode = FILE_SHARE_WRITE
00ED1191	8BE8	MOV EBP,EBX	Access = GENERIC_WRITE
00ED1193	68 00000040	PUSH 40000000	FileName = "C:\Users\IEUser\AppData\Local\Temp\MSVCR110.dll"
00ED1198	8D8424 5C0A00	LEA EAX,DWORD PTR SS:[ESP+85C]	CreateFileW
00ED1199	50	PUSH EAX	
00ED11A0	896C24 34	MOV DWORD PTR SS:[ESP+34],EBP	
00ED11A4	FFD6	CALL ESI	
00ED11A6	6A 00	PUSH 0	hTemplateFile = NULL
00ED11A8	68 80000000	PUSH 80	Attributes = NORMAL
00ED11AD	6A 02	PUSH 2	Mode = CREATE_ALWAYS
00ED11AF	6A 00	PUSH 0	pSecurity = NULL
00ED11B1	6A 02	PUSH 2	ShareMode = FILE_SHARE_WRITE
00ED11B3	68 00000040	PUSH 40000000	Access = GENERIC_WRITE
00ED11B8	8D8C24 4C0600	LEA ECX,DWORD PTR SS:[ESP+64C]	FileName = "C:\Users\IEUser\AppData\Local\Temp\adobe.exe"
00ED11BF	51	PUSH ECX	CreateFileW
00ED11C0	894424 30	MOV DWORD PTR SS:[ESP+30],EAX	
00ED11C4	FFD6	CALL ESI	

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).

00ED130C	8B1D FC80ED00	MOV EBX,DWORD PTR DS:[<&SHELL32.ShellExecuteW	SHELL32.ShellExecuteW
00ED1312	6A 00	PUSH 0	IsShown = 0
00ED1314	6A 00	PUSH 0	DefDir = NULL
00ED1316	6A 00	PUSH 0	Parameters = NULL
00ED1318	8D9424 380400	LEA EDX,DWORD PTR SS:[ESP+438]	FileName = "C:\Users\IEUser\AppData\Local\Temp\adobe.exe"
00ED1319	52	PUSH EDX	Operation = "open"
00ED1320	68 1893ED00	PUSH Position.00ED9318	hWnd = NULL
00ED1325	6A 00	PUSH 0	ShellExecuteW
00ED1327	FFD3	CALL EBX	
00ED1329	6A 00	PUSH 0	hTemplateFile = NULL
00ED132B	68 80000000	PUSH 80	Attributes = NORMAL
00ED132D	6A 02	PUSH 2	Mode = CREATE_ALWAYS
00ED132F	6A 00	PUSH 0	pSecurity = NULL
00ED1331	6A 02	PUSH 2	ShareMode = FILE_SHARE_WRITE
00ED1333	68 00000040	PUSH 40000000	Access = GENERIC_WRITE
00ED1334	8D9424 380100	LEA EDI,DWORD PTR SS:[ESP+138]	FileName = "C:\Users\IEUser\Downloads\Malware-sample\ActiveVersion\PositionRequirement-SeniorCivilEngineer.docx-53c1fd5ac99b569b278ffcc5a49
00ED1335	50	PUSH EDI	CreateFileW
00ED1337	FF15 0480ED00	CALL DWORD PTR DS:[<&KERNEL32.CreateFileW	
00ED1367	6A 01	PUSH 1	IsShown = 1
00ED1369	6A 00	PUSH 0	DefDir = NULL
00ED136B	6A 00	PUSH 0	Parameters = NULL
00ED136D	8D9424 2C0100	LEA EDI,DWORD PTR SS:[ESP+12C]	FileName = "C:\Users\IEUser\Downloads\Malware-sample\ActiveVersion\PositionRequirement-SeniorCivilEngineer.docx-53c1fd5ac99b569b278ffcc5a49
00ED136E	52	PUSH EDI	Operation = "open"
00ED1370	68 2493ED00	PUSH Position.00ED9324	hWnd = NULL
00ED1371	6A 00	PUSH 0	ShellExecuteW
00ED1373	FF15 0480ED00	CALL DWORD PTR DS:[<&SHELL32.ShellExecuteW	

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).

```
01011639 | . FF15 3480010 | CALL DWORD PTR DS:[<&KERNEL32.IsDebuggerPresent] | CKERNEL32.IsDebuggerPresent
0101163F | . A3 F06FE0 | MOV DWORD PTR DS:[1036FE0],EAX
KERNEL32.IsDebuggerPresent returned EAX = TRUE
EAX=011AFF34 (current registers)
[01036FE0]=0
```

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