

Shamoon v.2012

REVERSE ENGINEERING PROJECT FOR CSEC.202.03

ABDULMALIK K. BANASER

Contents

Environment Setup.....	2
Malware sample	2
Introduction.....	3
Analyzing the Malware.....	4
Basic Static Analysis.....	4
Summary	4
Analysis.....	4
Basic Dynamic Analysis	9
Summary	9
Analysis.....	9
Advance Static Analysis	13
Summary	13
Analysis.....	13
Advance Dynamic Analysis.....	27
Summary	27
Analysis.....	27
Summary	32
Resources	33

Environment Setup

For this project, I used an isolated Windows 10 Pro virtual machine, which is the safest way for reverse engineering. The host used was isolated meaning it has its networking options set to “host-Only Network,” which will prevent the malware to connect to an external network. For the tools, I installed variety of tools for basic static analysis, basic dynamic analysis, advance static analysis, and advance dynamic analysis.

Operating System	Windows 10 Pro
CPU	4 Core Processor
Ram	8 GB
Hard Disk	100 GB
Networking	Host-Only Network
Installed Software	<ul style="list-style-type: none">• Basic Static Tools• Basic Dynamic Tools• Advance Static Tools• Advance Dynamic Tools

Malware sample

For this project I used a malware sample called “Shamoon,” which is a ransomware malware that have targeted many hosts, and networks in the middle east.

Malware Name	Shamoon
Malware Version	2012
Malware MD5 Hash	fd7445210bc60baeeab77f69e1ba51b8
Malware SHA-256 Hash	a9f23dea34e3d3f8a84c513f950a53bd2c117640b202b0c65c7677d425ca02ec
Malware Download Link	https://github.com/ytisf/theZoo/tree/master/malwares/Binaries/Shamoon

Introduction

This project has given me the opportunity to reverse engineer a real-world example malware that have costs many companies millions of dollars. Also, this project allowed me to practice and sharpen my reversing skills that I have learned throughout this class. However, for this project I chose a malware sample that is called “Shamoon,” which is a ransomware that have targeted several locations and networks in middle east area. I chose this malware since my hometown was highly affected by this attack which happened in 2012. Different companies and individuals were infected by this malware such as the Saudi oil company Aramco, and Saudi Telecom Company (STC.) The main goal of this malware to hard-code and wipe workstation data and overwrite the data with corrupted images. The malware contains three major functionality which can be shown as the following:

- The Dropper

The dropper is the first functionality, and it is used to build a persistent service called NtsSrv on the infected device. It is available in 32-bit and 64-bit versions, and its payload is determined by the architecture it finds. It spreads its malicious code by copying itself to other networked machines.

- The Wiper

The wiper is Shamoon's next part, and it is used to release a third one. This is the Eldos driver, which overwrites the master boot record (MBR) on the hard disk with the malware's current picture. Without using the Window APIs, the driver allows user-mode access to the hard disk. The device would become unusable after the MBR has been overwritten.

- The Reporter

The Reporter is the final functionality that connects to a command-and-control server. This server is controlled by the attackers, who can use it to download more malware, adjust the pre-configured disk-wiping time, and submit reports to check that a certain disk has been deleted.

Finally, I picked this malware sample since the attack first appear in Saudi Arabia while my family's business was infected, so I found this opportunity is the perfect time with the knowledge I need to approach this malware sample.

Analyzing the Malware

This section will be divided to four sub sections which are “Basic Static Analysis,” “Basic Dynamic Analysis,” “Advance Static Analysis,” and “Advance Dynamic Analysis.”

Basic Static Analysis

Summary

In this section, I used basic static tools that helped me to analyze the malware sample without running it. This is an important part of malware analysis process which can show many host-based indicators, network-based indicators, and other valuable information about the malware. After this step, I was able to gain an overview about the malware’s functionality. The malware is appeared to be trying to manipulate the host’s file system, creating new services, and edit Windows’s registry keys.

Analysis

First, I used the “md5deep” tool to inspect the sample I have downloaded is the malware and have not been modified.

```
PS C:\Users\Windows\Desktop> md5deep.exe Shamoan.exe
md5deep.exe: WARNING: You are running a 32-bit program on a 64-bit system.
md5deep.exe: You probably want to use the 64-bit version of this program.
b14299fd4d1cbfb4cc7486d978398214 C:\Users\Windows\Desktop\Shamoan.exe
PS C:\Users\Windows\Desktop>
```

As we can see, we have the malware’s “MD5” hash, which can be used to check if the malware sample I installed is the malware itself and has not been modified. To compare this, I uploaded the malware sample I installed to “VirusTotal” and I got the same “MD5” hash.

58 / 69

58 security vendors flagged this file as malicious

4f02a9fcd2deb3936ede8f009bd08662bdb1f365c0f4a78b3757a98c2f40400
Distributed Link Tracking Server
direct-cpu-clock-access | power | via-tor

966.00 KB
Size

2021-04-23 06:21:07 UTC
13 days ago

EXE

DETECTION DETAILS RELATIONS BEHAVIOR COMMUNITY

Basic Properties

MD5	b14299fd4d1cbfb4cc7486d978398214
SHA-1	7c0dc6a8f4d2d762a07a523f19b7acd2258f7ecc
SHA-256	4f02a9fcd2deb3936ede8f009bd08662bdb1f365c0f4a78b3757a98c2f40400
Vhash	09506665d157510f8z62z4dz13z15z57z
Authenthash	6196a1f3d12f6dfd33332e4833f0d2ba9a6559520cb87ca28a4e471ab069315b
Imphash	da9452a2ec343ee77f6967d3524568
Rich PE header hash	5c41f15df17b778c689d8354c6c79b99
SSDEEP	12288:Xfz3ZXNpCwmGWCg98gJWGG2EbzXhK3qBUb7Ub:XfzZdE5Ng98gJWb2Ebzm3q
TLSH	T183256C2335F34372FEB7ED3480D25044AE4CB6E0DCD1BE0DEA865225A55A66CEC38796
File type	Win32 EXE
Magic	PE32 executable for MS Windows (console) Intel 80386 32-bit
TrID	Win32 Executable MS Visual C++ (generic) (48.8%)
TrID	Win64 Executable (generic) (16.4%)
TrID	Win32 Dynamic Link Library (generic) (10.2%)
TrID	Win16 NE executable (generic) (7.8%)
TrID	Win32 Executable (generic) (7%)
File size	966.00 KB (989184 bytes)

Second, after checking that I have downloaded the right executable, I used the “string” tool to look up for interesting strings that might give us a host-based indicators, network-based indicators, or other valuable information. Since we are analyzing a big malware, I tried to run the “string” tool with some “findstr” command to speed up the process of finding the information needed.

```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamoon.exe | findstr "c:"
c:\windows\temp\out17626867.txt
```

As we can see a “.txt” file that might be used as a host-based indicator.

```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamoon.exe | findstr ".cmd"
\System32\cmd.exe /c "ping -n 30 127.0.0.1 >nul && sc config TrkSvr binpath= system32\trksrv.exe && ping -n 10 127.0.0.1 >nul && sc start TrkSvr "
```

In the previous screenshot, we can see that the malware will try to test some network functionality, and config a service then starts the configured service.

```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamoon.exe | findstr "system"
C:\Windows\system32\svchost.exe -k netsvcs
\system32\kernel32.dll
\system32\
\system32\csrss.exe
\System32\cmd.exe /c "ping -n 30 127.0.0.1 >nul && sc config TrkSvr binpath= system32\trksrv.exe && ping -n 10 127.0.0.1 >nul && sc start TrkSvr "
system
Read-only file system
Too many open files in system
\system32\
..?AVsystem_error@std@
PS C:\Users\Windows\Desktop\Shamoon>
```

As we can see some other host-based indicators and some DLLs that is used by the malware itself.

```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamoon.exe | findstr ".com"
Sysinternals - www.sysinternals.com
Visual C++ CRT: Not enough memory to complete call to strerror.
This indicates a bug in your application. It is most likely the result of calling an MSIL-compiled (/clr) function from a native constructor or from DllMain.
testdomain.com
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamoon.exe | findstr "test"
test123
test456
test789
testdomain.com
PS C:\Users\Windows\Desktop\Shamoon>
```

In the previous step, I tried to find some network-based indicators, so I searched for a “.com” domain. After that, I found a “testdomain.com,” and then I searched for “test,” to see if I would find anything interesting.

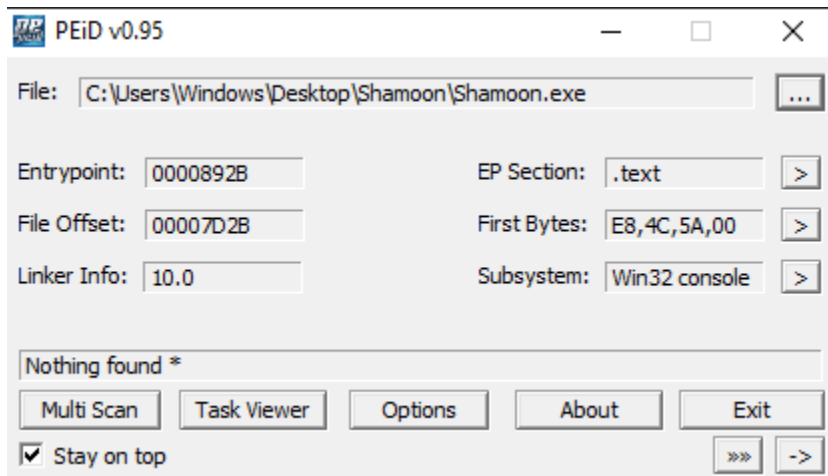
```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamoon.exe | findstr ".dll"
kernel32.dll
\system32\kernel32.dll
netapi32.dll
mscoree.dll
NETAPI32.dll
WS2_32.dll
KERNEL32.dll
USER32.dll
ADVAPI32.dll
SHELL32.dll
```

As we can see, I searched for “.dll” to examine what DLLs the malware will be using and understanding what the main functionality of the malware is.

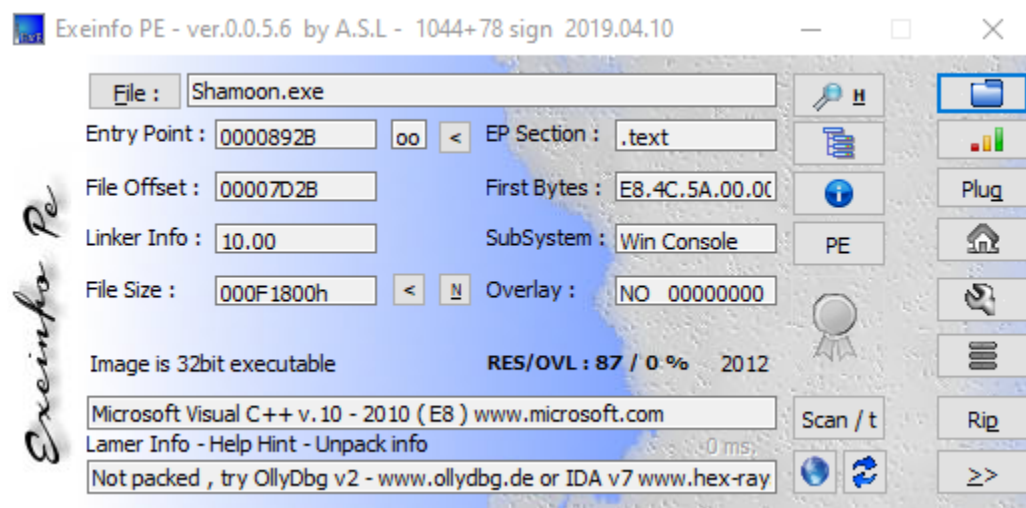
```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamoon.exe | findstr "Services"
SYSTEM\CurrentControlSet\Services\TrkSvr
PS C:\Users\Windows\Desktop\Shamoon>
```

Lastly, since I noticed the malware will run a service, so I checked for any new created services, which is also can be used as another host-based indicator.

Third, I tried to examine whether the malware is packed or not, and check other information such as creation time, time stamps, DLL calls. For this purpose, I used “PEiD, Exeinfo PE, and PEsview” tools.



As we can in the previous screenshot that the malware is not packed by checking the “EP Section,” but since this is not enough, I used “Exeinfo PE” to check the compiler used for this malware.



Then, I moved to check the DLL calls and imported functions to gain better understanding of the malware.

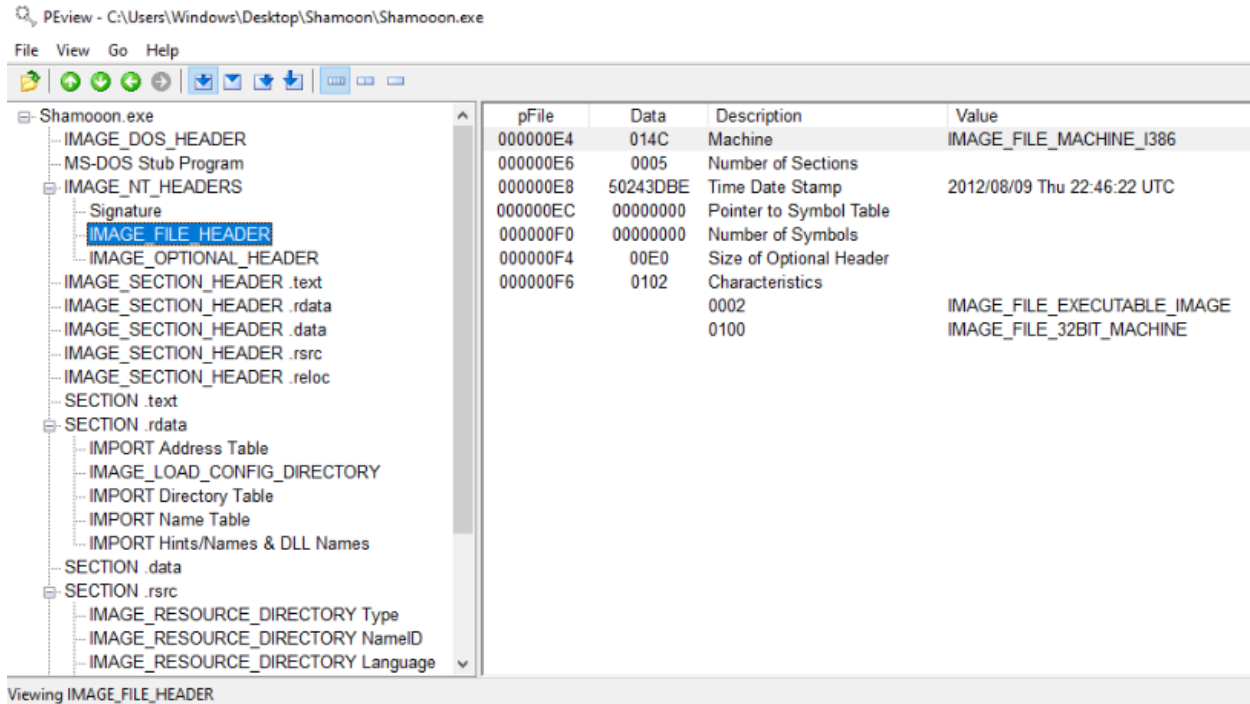
PEView - C:\Users\Windows\Desktop\Shamoon\Shamoon.exe

File View Go Help

	pFile	Data	Description	Value
Shamoon.exe				
IMAGE_DOS_HEADER	00014E00	0001B88A	Hint/Name RVA	02C8 StartServiceCtrlDispatcherW
MS-DOS Stub Program	00014E04	0001B88C	Hint/Name RVA	0288 RegisterServiceCtrlHandlerW
IMAGE_NT_HEADERS	00014E08	0001B8A8	Hint/Name RVA	02C0 SetServiceStatus
IMAGE_SECTION_HEADER .text	00014E0C	0001B876	Hint/Name RVA	026E RegQueryValueExW
IMAGE_SECTION_HEADER .rdata	00014E10	0001B864	Hint/Name RVA	01F9 OpenSCManagerW
IMAGE_SECTION_HEADER .data	00014E14	0001B854	Hint/Name RVA	01FB OpenServiceW
IMAGE_SECTION_HEADER .rsrc	00014E18	0001B83E	Hint/Name RVA	0224 QueryServiceConfigW
IMAGE_SECTION_HEADER .reloc	00014E1C	0001B826	Hint/Name RVA	0050 ChangeServiceConfigW
SECTION .text	00014E20	0001B810	Hint/Name RVA	0057 CloseServiceHandle
SECTION .rdata	00014E24	0001B7FE	Hint/Name RVA	0081 CreateServiceW
IMPORT Address Table	00014E28	0001B7E6	Hint/Name RVA	004E ChangeServiceConfig2W
IMAGE_LOAD_CONFIG_DIRECTORY	00014E2C	0001B7D6	Hint/Name RVA	0261 RegOpenKeyExW
IMPORT Directory Table	00014E30	0001B7C4	Hint/Name RVA	0248 RegDeleteValueW
IMPORT Name Table	00014E34	0001B7B6	Hint/Name RVA	0230 RegCloseKey
IMPORT Hints/Names & DLL Names	00014E38	0001B7A6	Hint/Name RVA	02C9 StartServiceW
SECTION .data	00014E3C	00000000	End of Imports	ADVAPI32.dll
SECTION .rsrc				
IMAGE_RESOURCE_DIRECTORY Type	00014E40	0001B5AC	Hint/Name RVA	0344 LocalAlloc
IMAGE_RESOURCE_DIRECTORY NameID	00014E44	0001B5BA	Hint/Name RVA	0202 GetLastError
IMAGE_RESOURCE_DIRECTORY Language	00014E48	0001B5CA	Hint/Name RVA	0360 MoveFileExW
IMAGE_RESOURCE_DATA_ENTRY	00014E4C	0001B5D8	Hint/Name RVA	00D6 DeleteFileW
IMAGE_RESOURCE_DIRECTORY_STRING	00014E50	0001B5E6	Hint/Name RVA	0245 GetProcAddress
PKCS12 0070 0000	00014E54	0001B5F8	Hint/Name RVA	0218 GetModuleHandleW
PKCS7 0071 0000	00014E58	0001B60C	Hint/Name RVA	0525 WriteFile
VERSION 0001 0000	00014E5C	0001B618	Hint/Name RVA	008F CreateFileW
X509 0074 0000	00014E60	0001B626	Hint/Name RVA	04B1 SizeofResource
SECTION .reloc				
	00014E64	0001B638	Hint/Name RVA	0354 LockResource
	00014E68	0001B648	Hint/Name RVA	0341 LoadResource
	00014E6C	0001B658	Hint/Name RVA	014E FindResourceW
	00014E70	0001B668	Hint/Name RVA	0187 GetCommandLineW
	00014E74	0001B67A	Hint/Name RVA	01F2 GetFileTime
	00014E78	0001B688	Hint/Name RVA	02AF GetWindowsDirectoryW
	00014E7C	0001B5A4	Hint/Name RVA	04B2 Sleep
	00014E80	0001B6AE	Hint/Name RVA	00B5 CreateThread

Viewing IMPORT Address Table

After examining the “IMPORT Address Table,” I was able to get an overview of the functionality of the malware, as I noticed many function calls that are used to manipulate the file system, process, services, and registries.

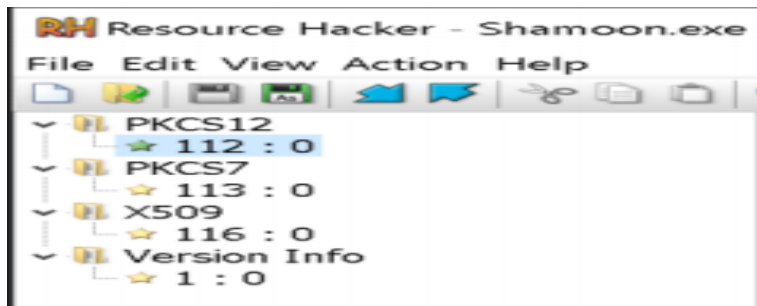


As we can see, I was able to check general information about the malware such as the time stamp, and compilation time. Then I compared the data I have with the data resulted back form “VirusTotal.”

History

Creation Time	2012-08-09 22:46:22
First Submission	2012-08-15 13:21:36
Last Submission	2021-01-04 06:47:20
Last Analysis	2021-04-23 06:21:07

Lastly, I used “resource hacker” to examine the resource that the malware’s using. Then I noticed data files that are recognized as resources, and the interesting part was that the resources meant to find GUI resource, not data files.



Basic Dynamic Analysis

Summary

In this section, I used basic dynamic tools that helped me to monitor the malware sample while running it. This is an important part of malware analysis process which can show many host-based indicators, network-based indicators, and other valuable information about the malware. After this step, I was able to examine the affected files, and newly created files and services. Also, I was able to notice the different in the host's registry keys as the malware have ran and implant itself to the system.

Analysis

First, I ran "Process Monitor" to monitor the process running by the malware. Additionally, before running the malware I started "Regshot" tool and took initial shot of the registries then a second shot to compare them and spot any new addition, modification, and deletion.

```
-----
Keys added: 35142
-----
HKLM\SYSTEM
HKLM\SYSTEM\ActivationBroker
HKLM\SYSTEM\ActivationBroker\Plugins
HKLM\SYSTEM\ActivationBroker\Plugins\{0913ACCF-B1AB-4EEE-A0C7-F4D7C12F4EEC}
HKLM\SYSTEM\ActivationBroker\Plugins\{14F3C12D-7712-42CC-B7CC-64D2B8560C43}
HKLM\SYSTEM\ActivationBroker\Plugins\{17821A1B-6C59-48E0-A448-68C9AD2C5BFE}
HKLM\SYSTEM\ActivationBroker\Plugins\{5672888A-BBF5-482E-B7B9-742C70C604D8}
HKLM\SYSTEM\ActivationBroker\Plugins\{8ED392B6-23C2-4C3C-9126-D12D68E621FD}
HKLM\SYSTEM\ActivationBroker\Plugins\{9CC1CC97-48C6-43DB-8265-48D9C8E192DD}
HKLM\SYSTEM\ActivationBroker\Plugins\{AA67AF38-4AE0-4B49-8A56-ADF78DBED45A}
HKLM\SYSTEM\ActivationBroker\Plugins\{AC59432D-8659-48C4-A584-AFEB920256F}
HKLM\SYSTEM\ActivationBroker\Plugins\{C2745EC3-CF23-4601-92EF-D189B711F933}
HKLM\SYSTEM\ActivationBroker\Plugins\{D6AC71F0-D4A7-41DD-88C4-B9985855D546}
HKLM\SYSTEM\ActivationBroker\Plugins\{F00006F2-44BC-44EF-808B-B26002A183C2}
HKLM\SYSTEM\ActivationBroker\Plugins\{F22D2A32-F1F4-4D62-AF5E-E5E8253AC6A6}
HKLM\SYSTEM\ActivationBroker\Plugins\{F48B770A-CBES-44C2-8D4F-931DE9CEE6FA}
HKLM\SYSTEM\ControlSet001
HKLM\SYSTEM\ControlSet001\Control
HKLM\SYSTEM\ControlSet001\Control\AccessibilitySettings
HKLM\SYSTEM\ControlSet001\Control\AccessibilitySettings\Theme
HKLM\SYSTEM\ControlSet001\Control\ACPI
HKLM\SYSTEM\ControlSet001\Control\AppID
```

Values modified: 38

[illegible]

Total changes: 142900

As we can see in the previous screenshot, the malware has done a total of 142900 changes to our registry keys. These changes can vary from addition to modification to deletion. Additionally, I was able to spot some registry keys that are used to manipulate a service and overwrite files.

Process Monitor - Sysinternals: www.sysinternals.com

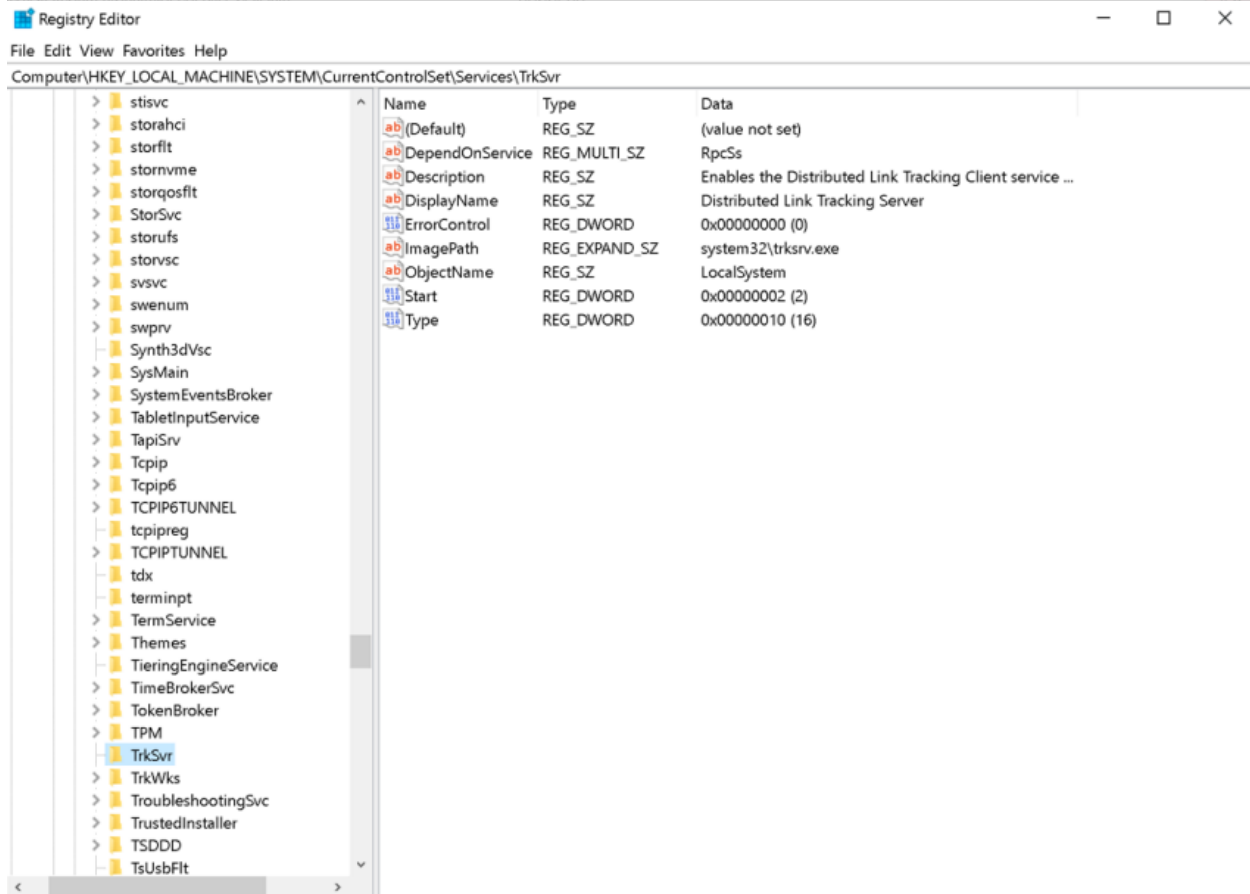
File Edit Event Filter Tools Options Help

Time ...	Process Name	PID	Operation	Path	Result	Detail
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\System\CurrentControlSet\Services\bindft\SupportedFeatures	SUCCESS	Type: REG_DWO...
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\bindft	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\PROCMON24	REPARSE	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\PROCMON24	SUCCESS	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\System\CurrentControlSet\Services\PROCMON24\SupportedFeatures	SUCCESS	Type: REG_DWO...
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\PROCMON24	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\WdFilter	REPARSE	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\WdFilter	SUCCESS	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\System\CurrentControlSet\Services\WdFilter\SupportedFeatures	SUCCESS	Type: REG_DWO...
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\WdFilter	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\Uafv	REPARSE	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\Uafv	SUCCESS	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\System\CurrentControlSet\Services\Uafv\SupportedFeatures	SUCCESS	Type: REG_DWO...
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\Uafv	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\Wof	REPARSE	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\Wof	SUCCESS	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\System\CurrentControlSet\Services\Wof\SupportedFeatures	SUCCESS	Type: REG_DWO...
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\Wof	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\FileInfo	REPARSE	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\FileInfo	SUCCESS	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\System\CurrentControlSet\Services\FileInfo\SupportedFeatures	SUCCESS	Type: REG_DWO...
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\FileInfo	SUCCESS	
1:58:0...	Shamoon.exe	1540	FileSystemControl	C:\Users\Windows\Desktop\Shamoon.exe	DEVICE FEATUR...	Control: FSCTL_O...
1:58:0...	Shamoon.exe	1540	ReadFile	C:\Users\Windows\Desktop\Shamoon.exe	SUCCESS	Offset: 0, Length: 2...
1:58:0...	Shamoon.exe	1540	WriteFile	C:\Windows\System32\trksvr.exe	SUCCESS	Offset: 0, Length: 2...
1:58:0...	Shamoon.exe	1540	ReadFile	C:\Users\Windows\Desktop\Shamoon.exe	SUCCESS	Offset: 262,144, Le...
1:58:0...	Shamoon.exe	1540	WriteFile	C:\Windows\System32\trksvr.exe	SUCCESS	Offset: 262,144, Le...
1:58:0...	Shamoon.exe	1540	ReadFile	C:\Users\Windows\Desktop\Shamoon.exe	SUCCESS	Offset: 524,288, Le...
1:58:0...	Shamoon.exe	1540	WriteFile	C:\Windows\System32\trksvr.exe	SUCCESS	Offset: 524,288, Le...
1:58:0...	Shamoon.exe	1540	ReadFile	C:\Users\Windows\Desktop\Shamoon.exe	SUCCESS	Offset: 786,432, Le...
1:58:0...	Shamoon.exe	1540	WriteFile	C:\Windows\System32\trksvr.exe	SUCCESS	Offset: 786,432, Le...
1:58:0...	Shamoon.exe	1540	SetBasicInform...	C:\Windows\System32\trksvr.exe	SUCCESS	CreationTime: 0, L...
1:58:0...	Shamoon.exe	1540	QueryRemotePr...	C:\Windows\System32\trksvr.exe	INVALID PARAM...	
1:58:0...	Shamoon.exe	1540	CloseFile	C:\Windows\System32\trksvr.exe	SUCCESS	
1:58:0...	Shamoon.exe	1540	CloseFile	C:\Users\Windows\Desktop\Shamoon.exe	SUCCESS	
1:58:0...	Shamoon.exe	1540	CreateFile	C:\Windows\System32\trksvr.exe	SUCCESS	Desired Access: G...
1:58:0...	Shamoon.exe	1540	SetBasicInform...	C:\Windows\System32\trksvr.exe	SUCCESS	CreationTime: 11/1...
1:58:0...	Shamoon.exe	1540	CloseFile	C:\Windows\System32\trksvr.exe	SUCCESS	
1:58:0...	Shamoon.exe	1540	ReadFile	C:\Windows\SysWOW64\eechost.dll	SUCCESS	Offset: 304,128, Le...
1:58:0...	Shamoon.exe	1540	RegQueryKey	HKLM	SUCCESS	Query: HandleTag...
1:58:0...	Shamoon.exe	1540	RegQueryKey	HKLM	SUCCESS	Query: Name
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\SYSTEM\CurrentControlSet\Services\TrkSvr	REPARSE	Desired Access: All...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\TrkSvr	SUCCESS	Desired Access: All...
1:58:0...	Shamoon.exe	1540	RegSetInfoKey	HKLM\System\CurrentControlSet\Services\TrkSvr	SUCCESS	KeySetInformation...
1:58:0...	Shamoon.exe	1540	RegDeleteValue	HKLM\System\CurrentControlSet\Services\TrkSvr\WOW64	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\TrkSvr	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\SOFTWARE\Microsoft\Idle	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM	SUCCESS	
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\Software\WOW6432Node\Microsoft\Windows NT\CurrentVersion\GRE_Initialize	REPARSE	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\GRE_Initialize	SUCCESS	Desired Access: R...
1:58:0...	Shamoon.exe	1540	RegSetInfoKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\GRE_Initialize	SUCCESS	KeySetInformation...
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\GRE_Initialize\DisableMetaFiles	NAME NOT FOUND	Length: 20
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\GRE_Initialize	SUCCESS	
1:58:0...	Shamoon.exe	1540	Thread Exit		SUCCESS	Thread ID: 6160, ...
1:58:0...	Shamoon.exe	1540	Thread Exit		SUCCESS	Thread ID: 860, Us...
1:58:0...	Shamoon.exe	1540	Thread Exit		SUCCESS	Thread ID: 8468, ...
1:58:0...	Shamoon.exe	1540	Process Exit		SUCCESS	Exit Status: 0, User...
1:58:0...	Shamoon.exe	1540	RegOpenKey	HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-1821375579-428763197...	SUCCESS	Desired Access: All...
1:58:0...	Shamoon.exe	1540	RegQueryValue	HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-1821375579-428763197...	SUCCESS	Type: REG_BINA...
1:58:0...	Shamoon.exe	1540	RegSetValue	HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-1821375579-428763197...	SUCCESS	Type: REG_BINA...
1:58:0...	Shamoon.exe	1540	RegCloseKey	HKLM\System\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-1821375579-428763197...	SUCCESS	

Showing 2,448 of 5,143,960 events (0.047%)

Backed up virtual memory

In the previous screenshot we can see that the malware has successfully opened a new registry key that is used to control and run a service called “TrkSvr,” which we previously seen during the basic static analysis.



After I noticed that the service has been created and executed by the malware, I went to double check that the service exists on our hosts, and I was able to spot the registry key added for it.

Advance Static Analysis

Summary

In this section, I used advance static tools that helped me to analyze the malware sample without running it. But this time we will be revising and viewing the actual code of the malware using “IDA PRO.” The limitation of this process is that we only can see the assembly instructions of the actual code, not the high-level language of the malware itself. After this process, I noticed different argument and hidden functionality of the malware such as arguments, Windows API function calls, and other functionalities. One of the main functionalities is that the malware will try to load a resource, then write that resource on the hard disk to wipe all the data.

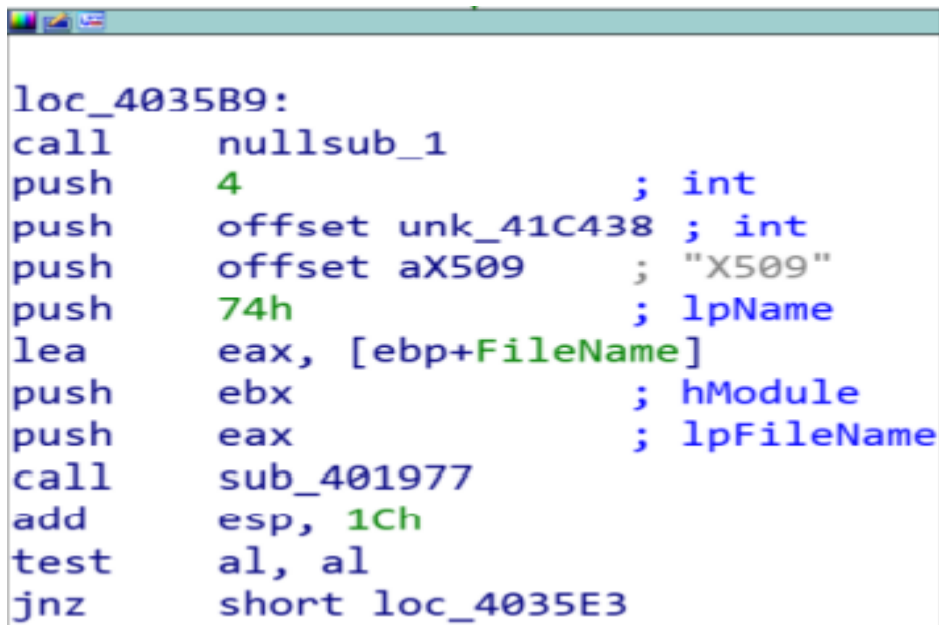
Resource summary:

- X509: determine how the data should be written to system32 directory.
- PKCS7: create a persistence method by creating a new service called “TrkSvr.”
- PKCS12: Wipe and overwrite the data on the hard disk with preloaded strings and corrupted images.

While I was analyzing the malware, I tried to gather as much as I can of the information to fully understand each one of the resources.

Analysis

First, searched for the resources we found during our basic static analysis, which are “PKCS7,” “X509,” and “PKCS12.” I found these to be interesting since they appeared to be data files so we might find something interesting in there.



```
loc_4035B9:
call    nullsub_1
push    4                ; int
push    offset unk_41C438 ; int
push    offset aX509     ; "X509"
push    74h              ; lpName
lea     eax, [ebp+FileName]
push    ebx              ; hModule
push    eax              ; lpFileName
call    sub_401977
add     esp, 1Ch
test    al, al
jnz     short loc_4035E3
```

```
loc_4033C9:                ; int
push      4
push      offset unk_41C434 ; int
push      offset Type      ; "PKCS7"
push      71h              ; lpName
lea       eax, [ebp+FileName]
push      0                ; hModule
push      eax              ; lpFileName
call      sub_401977
add       esp, 18h
test      al, al
jz        loc_403480
```

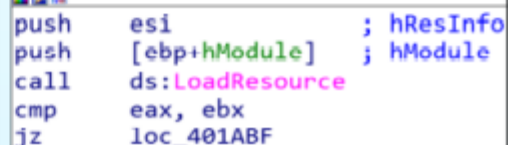
```
loc_4057E6:                ; int
push      4
push      offset unk_41C430 ; int
push      offset aPkcs12   ; "PKCS12"
push      70h              ; lpName
lea       eax, [ebp+FileName]
push      ebx              ; hModule
push      eax              ; lpFileName
call      sub_401977
add       esp, 18h
test      al, al
jz        short loc_405870
```


Then I found these strings unique, and I tried to see what other function might be using them. It appeared to me that all these strings are used to be pushed to a function as argument where we can see the function in the screenshot below.

```
sub_401977 proc near
```

```
lpAddress= dword ptr -20h
NumberOfBytesWritten= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
var_10= dword ptr -10h
hFile= dword ptr -0Ch
var_8= dword ptr -8
Buffer= byte ptr -1
lpFileName= dword ptr 8
hModule= dword ptr 0Ch
lpName= dword ptr 10h
lpType= dword ptr 14h
arg_10= dword ptr 18h
arg_14= dword ptr 1Ch
```

```
push    ebp
mov     ebp, esp
sub     esp, 20h
push    ebx
push    esi
push    [ebp+lpType]    ; lpType
push    [ebp+lpName]    ; lpName
push    [ebp+hModule]   ; hModule
call    ds:FindResourceW
mov     esi, eax
xor     ebx, ebx
cmp     esi, ebx
jz      loc_401ABF
```



```
push    esi                ; hResInfo
push    [ebp+hModule]      ; hModule
call    ds:LoadResource
cmp     eax, ebx
jz      loc_401ABF
```

Where in this function, the malware is trying to open each resource that is pushed as an argument using the API call to “FindResourceW,” and load that resource using the API call to “LoadResource.” Then I continued to trace the function and I noticed multiple if-statements that would take place if the zero flag were set.

```

push    eax                ; hResData
call    ds:LockResource
mov     [ebp+var_18], eax
cmp     eax, ebx
jz      loc_401ABF

push    esi                ; hResInfo
push    [ebp+hModule]      ; hModule
call    ds:SizeofResource
mov     [ebp+var_8], eax
lea     eax, [ebp+var_14]
push    eax
mov     [ebp+var_14], ebx
call    sub_401769
pop     ecx
push    ebx                ; hTemplateFile
push    ebx                ; dwFlagsAndAttributes
push    2                  ; dwCreationDisposition
push    ebx                ; lpSecurityAttributes
push    1                  ; dwShareMode
push    40000000h          ; dwDesiredAccess
push    [ebp+lpFileName] ; lpFileName
call    ds:CreateFileW
push    [ebp+var_14]
mov     [ebp+hFile], eax
call    sub_401792
cmp     [ebp+hFile], 0FFFFFFFh
pop     ecx
jz      loc_401ABF

push    edi
mov     edi, ds:WriteFile
xor     esi, esi
mov     [ebp+NumberOfBytesWritten], ebx

loc_401A0F:
cmp     esi, [ebp+var_8]
jnb     loc_401AB1

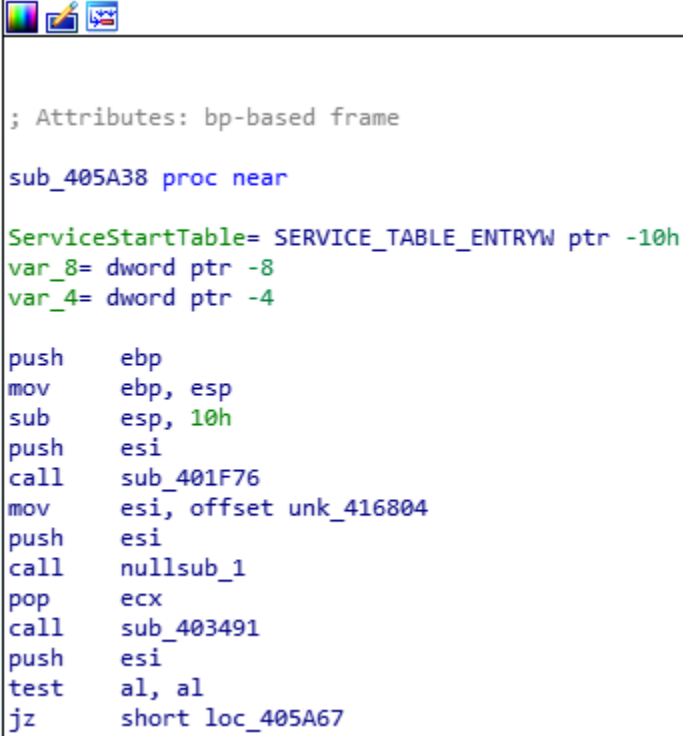
xor     edx, edx
mov     eax, esi
div     [ebp+arg_14]
mov     eax, [ebp+arg_10]
mov     ecx, [ebp+var_18]
push    ebx                ; lpOverlapped
mov     al, [edx+eax]
xor     al, [esi+ecx]
mov     [ebp+Buffer], al
lea     eax, [ebp+NumberOfBytesWritten]
push    eax                ; lpNumberOfBytesWritten
push    1                  ; nNumberOfBytesToWrite
lea     eax, [ebp+Buffer]
push    eax                ; lpBuffer
push    [ebp+hFile]        ; hFile
call    edi ; WriteFile
inc     esi
cmp     esi, 400h
jnb     short loc_401A0F

```

As we can see that there are other multiple API calls, and according to MSDN documentation, and the code structure found in the malware, I can predict the malware to try to load a resource (which is a big chunk of corrupted data,) then lock the resource so it can have a read/write access. After locking the resource, it will

get the size of the resource, then create a file of that size. After that it will write the file on the hard disk to wipe and corrupt the data.

Moving forward to the function starting at “SUB_405A38,” I noticed some other function calls, so I tried to inspect them to understand the main purpose of the function. However, the function calls at “SUB_405A67,” it will attempt to load an external path as the path of Windows’s kernel32.dll. Additionally, I noticed the behavior of this function is trying to parse the command line arguments passed using the API function calls of GetCommandLineW.



```
; Attributes: bp-based frame

sub_405A38 proc near

ServiceStartTable= SERVICE_TABLE_ENTRYW ptr -10h
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 10h
push    esi
call    sub_401F76
mov     esi, offset unk_416804
push    esi
call    nullsub_1
pop     ecx
call    sub_403491
push    esi
test    al, al
jz      short loc_405A67
```

```
loc_405A67:  
call    nullsub_1  
push    esi  
call    nullsub_1  
and     [ebp+var_8], 0  
and     [ebp+var_4], 0  
pop     ecx  
pop     ecx  
lea     eax, [ebp+ServiceStartTable]  
push    eax ; lpServiceStartTable  
mov     [ebp+ServiceStartTable.lpServiceName], offset aWow32 ; "wow32"  
mov     [ebp+ServiceStartTable.lpServiceProc], offset loc_405B50  
call    ds:StartServiceCtrlDispatcherW  
test    eax, eax  
jnz     short loc_405AAE
```

```
push    esi  
call    nullsub_1  
push    0  
call    sub_4058D0  
pop     ecx  
pop     ecx  
push    esi  
call    nullsub_1  
pop     ecx
```

```
loc_405AAE:  
call    sub_401230  
xor     eax, eax  
pop     esi  
leave  
retn  
sub_405A38 endp
```

However, I continued to go over the malware where I noticed another behavior of the malware where it will attempt to create and run a new service called “TrkSvr,” as we saw before in the basic dynamic analysis.

```
sub_403491 proc near

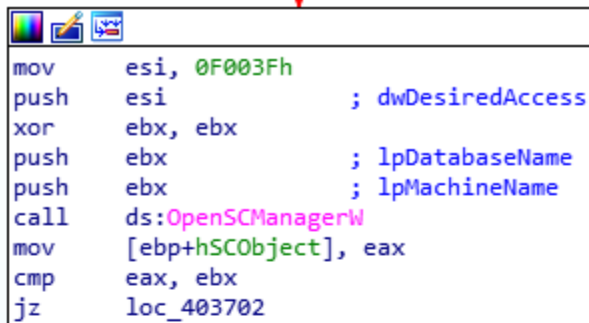
StartupInfo= _STARTUPINFO ptr -4D0h
ProcessInformation= _PROCESS_INFORMATION ptr -48Ch
hSCObject= dword ptr -47Ch
hService= dword ptr -478h
lpServiceConfig= dword ptr -474h
pcbBytesNeeded= dword ptr -470h
var_469= byte ptr -469h
CommandLine= word ptr -468h
FileName= word ptr -268h
var_68= dword ptr -68h
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 4D0h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
push    ebx
push    esi
push    edi
call    sub_4017BB
test    al, al
jz      loc_403702
```

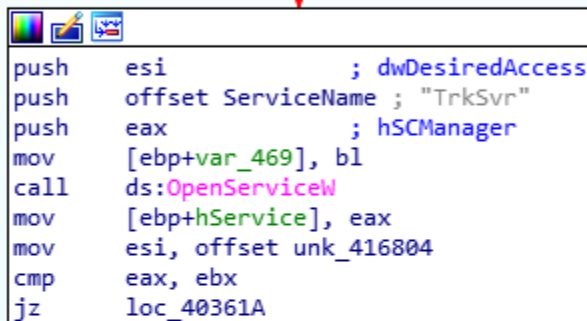
```
mov     esi, 0F003Fh
push    esi                ; dwDesiredAccess
xor     ebx, ebx
push    ebx                ; lpDatabaseName
push    ebx                ; lpMachineName
call    ds:OpenSCManagerW
mov     [ebp+hSCObject], eax
cmp     eax, ebx
jz      loc_403702
```

```
push    esi                ; dwDesiredAccess
push    offset ServiceName ; "TrkSvr"
push    eax                ; hSCManager
mov     [ebp+var_469], bl
call    ds:OpenServiceW
mov     [ebp+hService], eax
mov     esi, offset unk_416804
cmp     eax, ebx
jz      loc_40361A
```

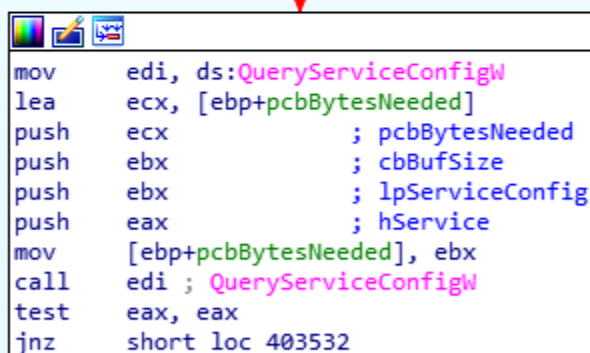
An additional behavior was discovered where going through the function at “SUB_403491,” I noticed that after creating the service and run it, the malware will attempt to check the config file of the service and check for any errors. If there is an error or a misconfigured file, the malware will be terminated, which can be found at “loc_403532.”



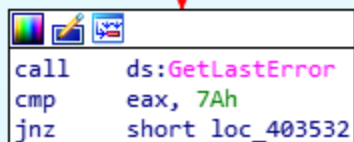
```
mov     esi, 0F003Fh
push    esi                ; dwDesiredAccess
xor     ebx, ebx
push    ebx                ; lpDatabaseName
push    ebx                ; lpMachineName
call    ds:OpenSCManagerW
mov     [ebp+hSCObject], eax
cmp     eax, ebx
jz      loc_403702
```



```
push    esi                ; dwDesiredAccess
push    offset ServiceName ; "TrkSvr"
push    eax                ; hSCManager
mov     [ebp+var_469], bl
call    ds:OpenServiceW
mov     [ebp+hService], eax
mov     esi, offset unk_416804
cmp     eax, ebx
jz      loc_40361A
```



```
mov     edi, ds:QueryServiceConfigW
lea     ecx, [ebp+pcbBytesNeeded]
push    ecx                ; pcbBytesNeeded
push    ebx                ; cbBufSize
push    ebx                ; lpServiceConfig
push    eax                ; hService
mov     [ebp+pcbBytesNeeded], ebx
call    edi ; QueryServiceConfigW
test    eax, eax
jnz     short loc_403532
```



```
call    ds:GetLastError
cmp     eax, 7Ah
jnz     short loc_403532
```

I noticed other few functions call. The first function calls appear to be a XOR decryption loop which I recognized while looking at “LOC_401A6F.”

Additionally, decryption functions, usually use a key to decrypt the data, so that key might be push as an argument to this function.

```

push    4                ; fIPProtect
push    3000h            ; flAllocationType
sub     eax, esi
push    eax              ; dwSize
push    ebx              ; lpAddress
call    ds:VirtualAlloc
mov     ecx, eax
mov     [ebp+lpAddress], ecx
cmp     ecx, ebx
jz      short loc_401AB1
    
```

```

mov     [ebp+var_10], ecx
sub     [ebp+var_10], esi
    
```

```

loc_401A6F:
xor     edx, edx
mov     eax, esi
div     [ebp+arg_14]
mov     eax, [ebp+arg_10]
mov     al, [edx+eax]
mov     edx, [ebp+var_18]
xor     al, [esi+edx]
mov     edx, [ebp+var_10]
mov     [edx+esi], al
inc     esi
cmp     esi, [ebp+var_8]
jnb     short loc_401A6F
    
```

```

push    ebx              ; lpOverlapped
lea     eax, [ebp+NumberOfBytesWritten]
push    eax              ; lpNumberOfBytesWritten
mov     eax, [ebp+var_8]
add     eax, 0FFFFFFC00h
push    eax              ; nNumberOfBytesToWrite
push    ecx              ; lpBuffer
push    [ebp+hFile]       ; hFile
call    edi ; WriteFile
push    8000h            ; dwFreeType
push    ebx              ; dwSize
push    [ebp+lpAddress]   ; lpAddress
call    ds:VirtualFree
    
```


After that I tried to locate the decryption key, I happened to find inside the resources' function. The function used for decryption can be found at either of "UNK_41C430," "UNK_41C438," and "UNK_41C434."

```
.data:0041C430 unk_41C430      db  25h ; %      |      ; DATA XREF: sub_4056B2+136fo
.data:0041C431              db  7Fh ;
.data:0041C432              db  5Dh ; ]
.data:0041C433              db  0FBh ; 0
```

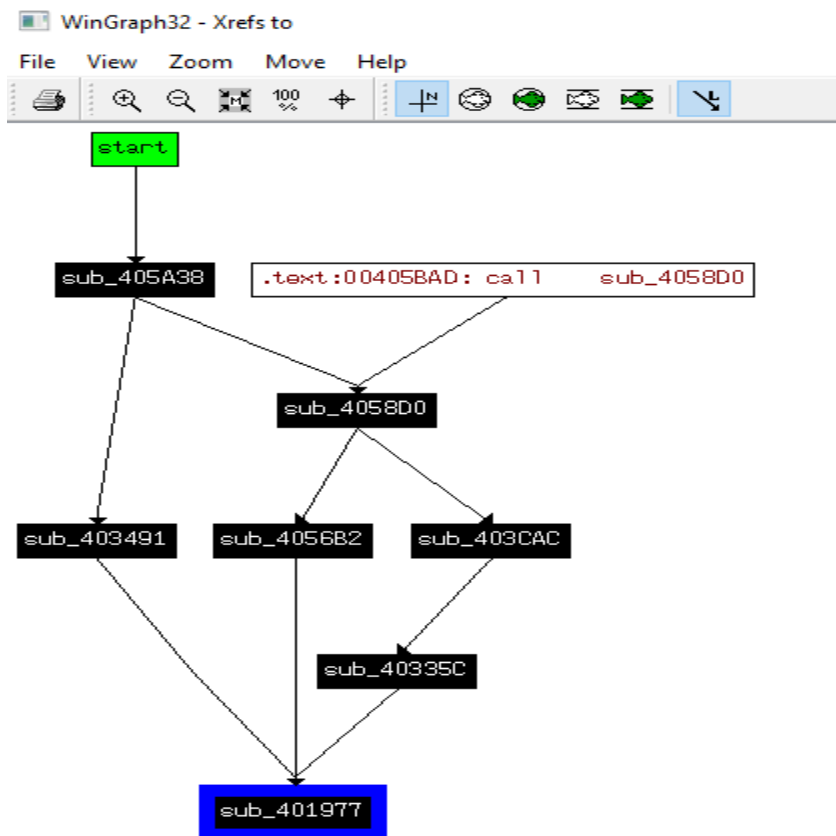
Which is called pushed as an argument inside all the resource's functions to other function that load and use the resources. Additionally, we can see that different resources will use different decryption keys

```
loc_4035B9:
call    nullsub_1
push    4 ; int
push    offset unk_41C438 ; int
push    offset ax509 ; ax509
push    74h ; lpName
lea     eax, [ebp+FileName]
push    ebx ; hModule
push    eax ; lpFileName
call    sub_401977
add     esp, 1Ch
test    al, al
jnz     short loc_4035E3
```

```
loc_4033C9: ; int
push    4
push    offset unk_41C434 ; int
push    offset type ; PRCS7
push    71h ; lpName
lea     eax, [ebp+FileName]
push    0 ; hModule
push    eax ; lpFileName
call    sub_401977
add     esp, 18h
test    al, al
jz      loc_403480
```

```
loc_4057E6:          ; int
push  4
push  offset unk_41C430 ; int
push  offset aPKCS12   ; PKCS12
push  70h              ; lpName
lea   eax, [ebp+FileName]
push  ebx              ; hModule
push  eax              ; lpFileName
call  sub_401977
add   esp, 18h
test  al, al
jz    short loc_405870
```

Additionally, I generated a graph to check my assumption whether all the resource will need to use the decryption function that is located at “SUB_401977.” So, I generated a graph stating from the decryption function to see what function are making calls to it.



However, we can see three functions that use the decryption function, and we have identified the function at “SUB_403491” as it will create a new service and run it.

Additionally, I noticed another function that will check the architecture of the system, then it will take different approach to write files to system32 directory.

```

mov     [ebp+edx*2+FileName], ax
lea     eax, [ebp+var_208]
xor     esi, esi
push    eax
mov     [ebp+var_208], esi
call    sub_401769
add     esp, 18h
push    esi                ; hTemplateFile
push    100000h            ; dwFlagsAndAttributes
push    3                  ; dwCreationDisposition
push    esi                ; lpSecurityAttributes
push    7                  ; dwShareMode
push    80000000h          ; dwDesiredAccess
lea     eax, [ebp+FileName]
push    eax                ; lpFileName
call    ds:CreateFileW
push    [ebp+var_208]
mov     edi, eax
call    sub_401792
pop     ecx
cmp     edi, 0FFFFFFFFh
jz      short loc_402063

```

```

push    offset LastWriteTime ; lpLastWriteTime
push    offset LastAccessTime ; lpLastAccessTime
push    offset CreationTime ; lpCreationTime
push    edi                ; hFile
call    ds:GetFileTime
test    eax, eax
jnz     short loc_40205C

```

```

mov     CreationTime.dwHighDateTime, esi
mov     CreationTime.dwLowDateTime, esi

```

```

loc_40205C:                ; hObject
push    edi
call    ds:CloseHandle

```

This step is needed by the malware so it can check how the service config file should be written. However, after this step the malware would have the service

installed and ready to use. But the malware has some code that will need to be run before it starts the service. First, it will try to test the creation of the service and check the networking stack if it has been corrupted or not. The test is shown below.

```
call    sub_4010AF
add     eax, eax
push    eax
lea     eax, [ebp+CommandLine]
push    edi
push    eax
call    sub_401050
push    offset aSystem32CmdExe ; "\\System32\\cmd.exe /c \"ping -n 30 127"...
call    sub_4010AF
add     esp, 18h
lea     eax, [eax+eax+2]
push    eax
push    offset aSystem32CmdExe ; "\\System32\\cmd.exe /c \"ping -n 30 127"...
push    edi
call    sub_4010AF
pop     ecx
lea     eax, [ebp+eax*2+CommandLine]
push    eax
call    sub_401050
push    44h
lea     eax, [ebp+StartupInfo]
push    ebx
push    eax
call    sub_407140
push    10h
lea     eax, [ebp+ProcessInformation]
push    ebx
push    eax
call    sub_407140
add     esp, 24h
lea     eax, [ebp+ProcessInformation]
push    eax ; lpProcessInformation
lea     eax, [ebp+StartupInfo]
push    eax ; lpStartupInfo
push    ebx ; lpCurrentDirectory
push    ebx ; lpEnvironment
push    8000000h ; dwCreationFlags
push    ebx ; bInheritHandles
push    ebx ; lpThreadAttributes
push    ebx ; lpProcessAttributes
lea     eax, [ebp+CommandLine]
push    eax ; lpCommandLine
push    ebx ; lpApplicationName
call    ds:CreateProcessW
test    eax, eax
jz      short loc_403702
```

If the malware has installed and configured the malware properly, then it will start a service for it, as we can see in the API call of “CreateProcessW.”

Another functionality of the malware is recognized while looking at “PKCS12,” we can see a function call at “SUB_4056B2” where in this function we have some hard coded strings, reference to a file on the file system, and an image file. My assumption to this function is that it is trying to overwrite the hard disk to wipe the data and corrupt the system.

```
mov     [ebp+var_4], ebx
mov     [ebp+var_218], offset aKijjjjnsnjbnn ; "kijjjjnsnjbnnbkbkjadcr\nkjsdjbhjsdbh"...
call    AsciiToWide
mov     eax, [ebp+var_228]
push    dword ptr [eax]
push    offset NewImageString
call    WideStrCpy
push    eax
call    sub_4051B3
add     esp, 0Ch
push    1
lea     ecx, [ebp+var_2E0]
call    sub_4055D8
push    40h
push    1
push    offset aCWindowsTempOu ; "c:\\windows\\temp\\out17626867.txt"
lea     ecx, [ebp+var_2E0]
mov     byte ptr [ebp+var_4], 1
call    ReadFile??
cmp     [ebp+var_274], ebx
jz      short loc_405771
```

```
loc_405870:                ; fuLoad
push    1
push    ebx                ; cy
push    ebx                ; cx
push    ebx                ; type
push    offset name        ; "myimage12767"
push    ebx                ; hInst
call    ds:LoadImageW
cmp     eax, ebx
jnz     short loc_4058B6
```

Advance Dynamic Analysis

Summary

In this section, I used advance dynamic tools that helped me to analyze the malware sample while running it and debugging. This is an important part of malware analysis process which can reveal the detailed functionality of the malware. After this step, I was able to gain almost full understanding of the malware's functionality.

Analysis

First, I checked the DLL imports to check if there are any runtime linking to the malware sample we are using. I found a similar output of what we found using basic static analysis.

Base	Module	Party	Path
00880000	shamoon.exe	User	C:\Users\Windows\Desktop\Shamoon.exe
6DAA0000	schedcli.dll	System	C:\Windows\SysWOW64\schedcli.dll
736F0000	srvccli.dll	System	C:\Windows\SysWOW64\srvccli.dll
74EA0000	apphelp.dll	System	C:\Windows\SysWOW64\apphelp.dll
75000000	ntmarta.dll	System	C:\Windows\SysWOW64\ntmarta.dll
751F0000	netutils.dll	System	C:\Windows\SysWOW64\netutils.dll
75220000	netapi32.dll	System	C:\Windows\SysWOW64\netapi32.dll
75C10000	kernel32.dll	System	C:\Windows\SysWOW64\kernel32.dll
75DC0000	shcore.dll	System	C:\Windows\SysWOW64\SHCore.dll
75E50000	ws2_32.dll	System	C:\Windows\SysWOW64\ws2_32.dll
75EC0000	msvcprt4.dll	System	C:\Windows\SysWOW64\msvcprt4.dll
76380000	rpcrt4.dll	System	C:\Windows\SysWOW64\rpcrt4.dll
764A0000	imm32.dll	System	C:\Windows\SysWOW64\imm32.dll
765D0000	sechost.dll	System	C:\Windows\SysWOW64\sechost.dll
76A40000	advapi32.dll	System	C:\Windows\SysWOW64\advapi32.dll
76AC0000	user32.dll	System	C:\Windows\SysWOW64\user32.dll
76C60000	kernelbase.dll	System	C:\Windows\SysWOW64\KernelBase.dll
76F10000	combase.dll	System	C:\Windows\SysWOW64\combase.dll
771A0000	win32u.dll	System	C:\Windows\SysWOW64\win32u.dll
77200000	gdi32.dll	System	C:\Windows\SysWOW64\gdi32.dll
77230000	gdi32full.dll	System	C:\Windows\SysWOW64\gdi32full.dll
773A0000	shell32.dll	System	C:\Windows\SysWOW64\shell32.dll
77960000	msvcrt.dll	System	C:\Windows\SysWOW64\msvcrt.dll
77B10000	ucrtbase.dll	System	C:\Windows\SysWOW64\ucrtbase.dll
77C40000	ntdll.dll	System	C:\Windows\SysWOW64\ntdll.dll

After checking the DLLs, I looked for the "EntryPoint," as it will lead us to the main function so we can see the true functionality of the malware.

0088890A	837D E4 00	cmp dword ptr ss:[ebp-1C],0	
0088890E	75 06	jne shamoon.888916	
00888910	50	push eax	
00888911	E8 39FEFFFF	call shamoon.88874F	
00888916	E8 59FEFFFF	call shamoon.888774	
00888918	C745 FC FFFFFFFF	mov dword ptr ss:[ebp-4],FFFFFFFF	
00888922	8B45 E0	mov eax,dword ptr ss:[ebp-20]	
00888925	E8 DB380000	call shamoon.88C205	
0088892A	C3	ret	
0088892B	E8 4C5A0000	call shamoon.88E37C	EntryPoint
00888930	E9 95FEFFFF	jmp shamoon.8887CA	
00888935	8BFF	mov edi,edi	
00888937	55	push ebp	
00888938	8BEC	mov ebp,esp	
0088893A	8B45 08	mov eax,dword ptr ss:[ebp+8]	

After accessing the malware's main function and go through the code, I found out the actual instructions that is used to create the "TrkSvr" that we found earlier. Second, I was able to see the absolute bath of the executable created by the malware of that specific service.

Log
Notes
Breakpoints
Memory Map
Call Stack
SEH
Script
Symbols
Source
References
Threads
Handles
Trace

```

00881288 55 push ebp
00881289 8BEC mov ebp,esp
0088128B 81EC 04040000 sub esp,404
00881291 A1 00CE8900 mov eax,dword ptr ds:[89CE00]
00881296 33C5 xor eax,ebp
00881298 8945 FC mov dword ptr ss:[ebp-4],eax
0088129B 8B45 08 mov eax,dword ptr ss:[ebp+8]
0088129E 8B4D 0C mov ecx,dword ptr ss:[ebp+C]
008812A1 53 push ebx
008812A2 56 push esi
008812A3 BB 3F000F00 mov ebx,F00F3F
008812A8 53 push ebx
008812A9 33F6 xor esi,esi
008812AB 56 push esi
008812AC 56 push eax
008812AD 89B5 FCFBFFFF mov dword ptr ss:[ebp-404],eax
008812B3 89B0 04FCFFFF mov dword ptr ss:[ebp-3FC],ecx
008812B9 FF15 10608900 call dword ptr ds:[<OpenServiceW>]
008812BF 89B5 00FCFFFF mov dword ptr ss:[ebp-400],eax
008812C5 3BC6 cmp eax,esi
008812C7 75 07 jne shamoon.8812D0
008812C9 32C0 xor al,al
008812CB 75 40 jne shamoon.881614
008812D0 57 push edi
008812D1 53 push ebx
008812D2 BF 80658900 mov edi,shamoon.8965B0
008812D7 57 push edi
008812D8 50 push eax
008812D9 FF15 14608900 call dword ptr ds:[<OpenServiceW>]
008812DF 89B5 10FCFFFF mov dword ptr ss:[ebp-3F0],eax
008812E5 3BC6 cmp eax,esi
008812E7 75 04 jne shamoon.8813D6
008812ED 8D80 lea ecx,dword ptr ss:[ebp-3F4]
008812F3 51 push ecx
008812F4 56 push esi
008812F5 56 push esi
008812F6 50 push eax
008812F7 89B5 0CFCFFFF mov dword ptr ss:[ebp-3F4],esi
008812FD FF15 18608900 call dword ptr ds:[<QueryServiceConfig>]
00881303 85C0 test eax,eax
00881305 75 1E jne shamoon.881325
00881307 FF15 44608900 call dword ptr ds:[<GetLastError>]
0088130D 83F8 7A cmp eax,7A
00881310 75 13 jne shamoon.881325
00881312 FF85 0CFCFFFF push dword ptr ss:[ebp-3F4]
00881318 56 push esi
00881319 FF15 40608900 call dword ptr ds:[<LocalAlloc>]
0088131F 89B5 08FCFFFF mov dword ptr ss:[ebp-3F8],eax
00881325 8D85 lea eax,dword ptr ss:[ebp-3F4]
00881328 50 push eax
00881329 89B5 0CFCFFFF mov dword ptr ss:[ebp-3F4],eax

```

[ebp+C]:L"C:\\windows\\system32\\trksvr.exe"

[ebp-3FC]:L"C:\\windows\\system32\\trksvr.exe"

edi:L"TrkSvr"

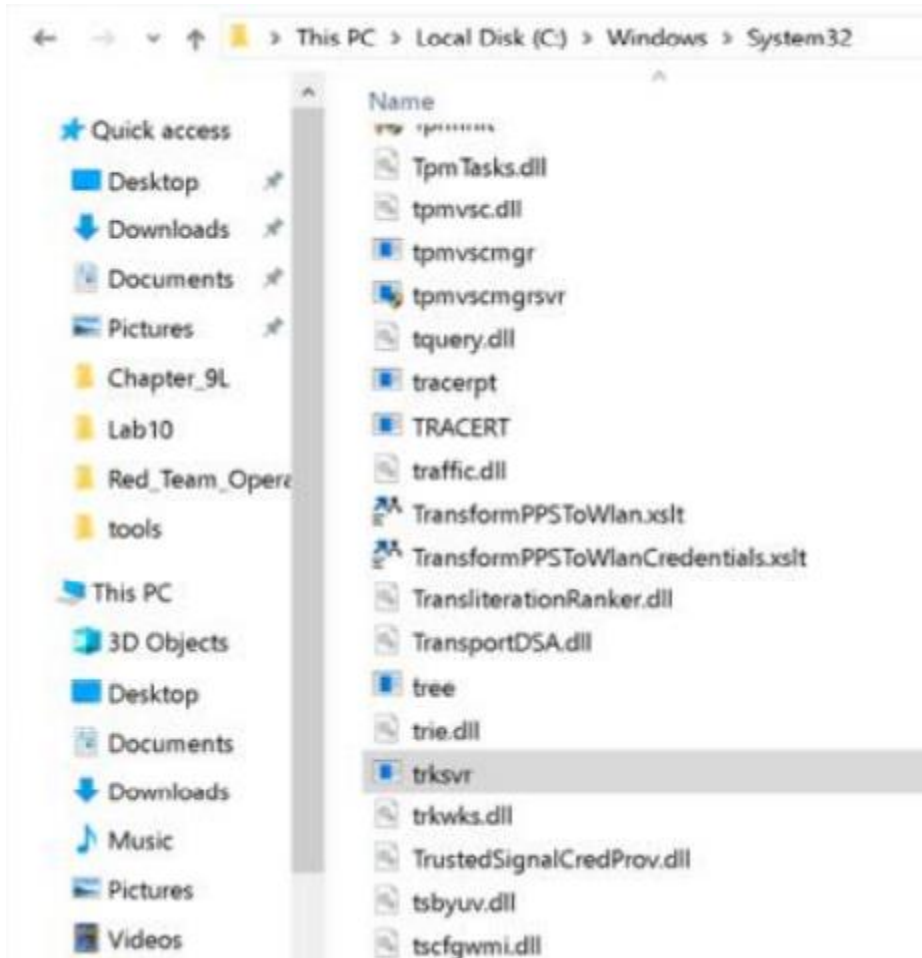
edi:L"TrkSvr", 8965B0:L"TrkSvr"

edi:L"TrkSvr"

7A: 'z'

However, after step over few times, I was able to see the actual executable file, and the registry key added. This indicate that the malware is running as we expected.

[illegible]



Third, I patched the malware to see what changes will happen in its behavior. To do so, I have changed one of the if-statement instructions to return a failure value so the malware would fail. The instruction I changed was the output of the “ping” command we inspected earlier in advance static analysis. When I patched that function, I noticed a weird behavior of the malware, but eventually it will crash. So, I guess there is nothing interesting in there.

008888D8	8945 E0	mov dword ptr ss:[ebp-20],eax
008888DB	3975 E4	cmp dword ptr ss:[ebp-1C],esi
008888DE	75 06	jne shamoon.8888E6
008888E1	E8 53FFFFFF	call shamoon.888739
008888E4	EB 2AFFFFFF	call shamoon.888765
008888E8	EB 2E	jmp shamoon.888918
008888ED	8845 EC	mov eax,dword ptr ss:[ebp-14]
008888F0	8808	mov ecx,dword ptr ds:[eax]
008888F2	8809	mov ecx,dword ptr ds:[ecx]
008888F4	894D DC	mov dword ptr ss:[ebp-24],ecx
008888F7	50	push eax
008888F8	51	push ecx
008888F9	E8 FC550000	call shamoon.88DEFA
008888FE	59	pop ecx
008888FF	59	pop ecx
00888900	C3	ret
00888901	8865 E8	mov esp,dword ptr ss:[ebp-18]
00888904	8845 DC	mov eax,dword ptr ss:[ebp-24]
00888907	8945 E0	mov dword ptr ss:[ebp-20],eax
0088890A	837D E4 00	cmp dword ptr ss:[ebp-1C],0
0088890E	75 06	jne shamoon.888916
00888910	50	push eax
00888911	E8 39FFFFFF	call shamoon.88874F
00888916	E8 59FFFFFF	call shamoon.888774
0088891B	C745 FC FFFFFFFF	mov dword ptr ss:[ebp-4],FFFFFFFF
00888922	8845 E0	mov eax,dword ptr ss:[ebp-20]

Fourth, I was able to spot the tracking service that was implanted by the malware and examine how the monitoring process goes. The tracking service is beaconing out to an external command-and-control server that is owned by the attacker. The attackers used this tracking service to make sure that the data has been completely corrupted by their malware.

00881390	6A 02	push 2	
00881392	6A 10	push 10	
00881394	FFB5 04CFFFFF	push dword ptr ss:[ebp-34]	
0088139A	FF15 1C608900	call dword ptr ds:[6CChangeServiceConf]	
008813A0	8B85 04CFFFFF	lea eax,dword ptr ss:[ebp-3C]	
008813A6	50	push eax	
008813A7	6A 02	push 2	
008813A9	FFB5 04CFFFFF	push dword ptr ss:[ebp-34]	
008813AF	C785 04CFFFFF	mov dword ptr ss:[ebp-3C],shamoon.8963	8963581L"Enables the Distributed Link Tracking Client service within the same domain to provide more reliable and efficient maintenance of links with
008813B9	E8 77	jmp shamoon.88145A	
008813BB	FFB5 04CFFFFF	push dword ptr ss:[ebp-34]	
008813C1	8B85 04CFFFFF	mov esi,dword ptr ds:[6CChangeServiceConf]	
008813C7	FFD6	call esi	
008813C9	FFB5 04CFFFFF	push dword ptr ss:[ebp-34]	
008813CF	FFD6	call esi	
008813D1	EB 10	jmp shamoon.881411	
008813D6	FF15 1C608900	call dword ptr ds:[6CChangeServiceConf]	
008813DC	FF15 1C608900	call dword ptr ds:[6CChangeServiceConf]	
008813E2	8D 24040000	cmp eax,4	
008813E7	0F85 18020000	jne shamoon.881405	
008813ED	56	push esi	
008813EE	56	push esi	
008813EF	68 44588900	push shamoon.896544	8965441L"Rpcss"
008813F4	56	push esi	
008813F5	56	push esi	
008813F6	56	push esi	
008813F7	56	push esi	
008813F8	56	push esi	
008813F9	56	push esi	
008813FA	56	push esi	
008813FB	56	push esi	
008813FC	56	push esi	
008813FD	56	push esi	
008813FE	56	push esi	
008813FF	56	push esi	
00881400	56	push esi	
00881401	56	push esi	
00881402	56	push esi	
00881403	56	push esi	
00881404	56	push esi	
00881405	56	push esi	
00881406	56	push esi	
00881407	56	push esi	
00881408	56	push esi	
00881409	56	push esi	
0088140A	56	push esi	
0088140B	56	push esi	
0088140C	56	push esi	
0088140D	56	push esi	
0088140E	56	push esi	
0088140F	56	push esi	
00881410	56	push esi	
00881411	56	push esi	
00881412	56	push esi	
00881413	56	push esi	
00881414	56	push esi	
00881415	56	push esi	
00881416	56	push esi	
00881417	56	push esi	
00881418	56	push esi	
00881419	56	push esi	
0088141A	56	push esi	
0088141B	56	push esi	
0088141C	56	push esi	
0088141D	56	push esi	
0088141E	56	push esi	
0088141F	56	push esi	
00881420	56	push esi	
00881421	56	push esi	
00881422	56	push esi	
00881423	56	push esi	
00881424	56	push esi	
00881425	56	push esi	
00881426	56	push esi	
00881427	56	push esi	
00881428	56	push esi	
00881429	56	push esi	
0088142A	56	push esi	
0088142B	56	push esi	
0088142C	56	push esi	
0088142D	56	push esi	
0088142E	56	push esi	
0088142F	56	push esi	
00881430	56	push esi	
00881431	56	push esi	
00881432	56	push esi	
00881433	56	push esi	
00881434	56	push esi	
00881435	56	push esi	
00881436	56	push esi	
00881437	56	push esi	
00881438	56	push esi	
00881439	56	push esi	
0088143A	56	push esi	
0088143B	56	push esi	
0088143C	56	push esi	
0088143D	56	push esi	
0088143E	56	push esi	
0088143F	56	push esi	
00881440	56	push esi	
00881441	56	push esi	
00881442	56	push esi	
00881443	56	push esi	
00881444	56	push esi	
00881445	56	push esi	
00881446	56	push esi	
00881447	56	push esi	
00881448	56	push esi	
00881449	56	push esi	
0088144A	56	push esi	
0088144B	56	push esi	
0088144C	56	push esi	
0088144D	56	push esi	
0088144E	56	push esi	
0088144F	56	push esi	
00881450	56	push esi	
00881451	56	push esi	
00881452	56	push esi	
00881453	56	push esi	
00881454	56	push esi	
00881455	56	push esi	
00881456	56	push esi	
00881457	56	push esi	
00881458	56	push esi	
00881459	56	push esi	
0088145A	56	push esi	
0088145B	56	push esi	
0088145C	56	push esi	
0088145D	56	push esi	
0088145E	56	push esi	
0088145F	56	push esi	
00881460	56	push esi	
00881461	56	push esi	
00881462	56	push esi	
00881463	56	push esi	
00881464	56	push esi	
00881465	56	push esi	
00881466	56	push esi	
00881467	56	push esi	
00881468	56	push esi	
00881469	56	push esi	
0088146A	56	push esi	
0088146B	56	push esi	
0088146C	56	push esi	
0088146D	56	push esi	
0088146E	56	push esi	
0088146F	56	push esi	
00881470	56	push esi	
00881471	56	push esi	
00881472	56	push esi	
00881473	56	push esi	
00881474	56	push esi	
00881475	56	push esi	
00881476	56	push esi	
00881477	56	push esi	
00881478	56	push esi	
00881479	56	push esi	
0088147A	56	push esi	
0088147B	56	push esi	
0088147C	56	push esi	
0088147D	56	push esi	
0088147E	56	push esi	
0088147F	56	push esi	
00881480	56	push esi	
00881481	56	push esi	
00881482	56	push esi	
00881483	56	push esi	
00881484	56	push esi	
00881485	56	push esi	
00881486	56	push esi	
00881487	56	push esi	
00881488	56	push esi	
00881489	56	push esi	
0088148A	56	push esi	
0088148B	56	push esi	
0088148C	56	push esi	
0088148D	56	push esi	
0088148E	56	push esi	
0088148F	56	push esi	
00881490	56	push esi	
00881491	56	push esi	
00881492	56	push esi	
00881493	56	push esi	
00881494	56	push esi	
00881495	56	push esi	
00881496	56	push esi	
00881497	56	push esi	
00881498	56	push esi	
00881499	56	push esi	
0088149A	56	push esi	
0088149B	56	push esi	
0088149C	56	push esi	
0088149D	56	push esi	
0088149E	56	push esi	
0088149F	56	push esi	
008814A0	56	push esi	
008814A1	56	push esi	
008814A2	56	push esi	
008814A3	56	push esi	
008814A4	56	push esi	
008814A5	56	push esi	
008814A6	56	push esi	
008814A7	56	push esi	
008814A8	56	push esi	
008814A9	56	push esi	
008814AA	56	push esi	
008814AB	56	push esi	
008814AC	56	push esi	
008814AD	56	push esi	
008814AE	56	push esi	
008814AF	56	push esi	
008814B0	56	push esi	
008814B1	56	push esi	
008814B2	56	push esi	
008814B3	56	push esi	
008814B4	56	push esi	
008814B5	56	push esi	
008814B6	56	push esi	
008814B7	56	push esi	
008814B8	56	push esi	
008814B9	56	push esi	
008814BA	56	push esi	
008814BB	56	push esi	
008814BC	56	push esi	
008814BD	56	push esi	
008814BE	56	push esi	
008814BF	56	push esi	
008814C0	56	push esi	
008814C1	56	push esi	
008814C2	56	push esi	
008814C3	56	push esi	
008814C4	56	push esi	
008814C5	56	push esi	
008814C6	56	push esi	
008814C7	56	push esi	
008814C8	56	push esi	
008814C9	56	push esi	
008814CA	56	push esi	
008814CB	56	push esi	
008814CC	56	push esi	
008814CD	56	push esi	
008814CE	56	push esi	
008814CF	56	push esi	
008814D0	56	push esi	
008814D1	56	push esi	
008814D2	56	push esi	
008814D3	56	push esi	
008814D4	56	push esi	
008814D5	56	push esi	
008814D6	56	push esi	
008814D7	56	push esi	
008814D8	56	push esi	
008814D9	56	push esi	
008814DA	56	push esi	
008814DB	56	push esi	

Fifth, I noticed the instructions the is responsible to determine which architecture to use while adding file to system32 directory.

00881761	5B	pop ebx	
00881762	E8 535A0000	call shamoon.88718A	
00881767	C9	leave	
00881768	C3	ret	
00881769	55	push ebp	
0088176A	8BEC	mov ebp,esp	
0088176C	68 E8658900	push shamoon.8965E8	8965E8: "Wow64DisableWow64FsRedirection"
00881771	68 CC658900	push shamoon.8965CC	8965CC: L"kernel32.dll"
00881776	FF15 54608900	call dword ptr ds:[<&GetModuleHandle>]	
0088177C	50	push eax	
0088177D	FF15 50608900	call dword ptr ds:[<&GetProcAddress>]	
00881783	85C0	test eax,eax	
00881785	74 07	je shamoon.88178E	
00881787	FF75 08	push dword ptr ss:[ebp+8]	
0088178A	FFD0	call eax	
0088178C	5D	pop ebp	
0088178D	C3	ret	
0088178E	33C0	xor eax,eax	
00881790	5D	pop ebp	
00881791	C3	ret	
00881792	55	push ebp	
00881793	8BEC	mov ebp,esp	
00881795	68 08668900	push shamoon.896608	896608: "Wow64RevertWow64FsRedirection"
0088179A	68 CC658900	push shamoon.8965CC	8965CC: L"kernel32.dll"
0088179F	FF15 54608900	call dword ptr ds:[<&GetModuleHandle>]	
008817A5	50	push eax	
008817A6	FF15 50608900	call dword ptr ds:[<&GetProcAddress>]	
008817AC	85C0	test eax,eax	
008817AE	74 07	je shamoon.881787	
00881780	FF75 08	push dword ptr ss:[ebp+8]	
00881783	FFD0	call eax	
00881785	5D	pop ebp	
00881786	C3	ret	
00881787	33C0	xor eax,eax	
00881789	5D	pop ebp	
0088178A	C3	ret	

Additionally, I was able to note the absolute path of the session created by the installed service. This session is used to transfer information between the victim and the attacker.

00881788	55	push ebp	
0088178C	8BEC	mov ebp,esp	
0088178E	81EC DC000000	sub esp,DC	
008817C4	A1 00CE8900	mov eax,dword ptr ds:[89CE00]	
008817C9	33C5	xor eax,ebp	
008817CB	8945 FC	mov dword ptr ss:[ebp-4],eax	
008817CE	53	push ebx	
008817CF	8D85 28FFFFFF	lea eax,dword ptr ss:[ebp-D8]	
008817D5	50	push eax	
008817D6	68 19000200	push 20019	
008817D8	33D8	xor ebx,ebx	
008817DD	53	push ebx	
008817DE	68 70668900	push shamoon.896670	896670: L"SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment"
008817E3	68 02000000	push 80000002	
008817E8	FF15 2C608900	call dword ptr ds:[<&RegOpenKeyEx>]	
008817EE	85C0	test eax,eax	
008817F0	74 07	jne shamoon.881895	
008817F6	8D85 2CFFFFFF	lea eax,dword ptr ss:[ebp-D4]	
008817FC	50	push eax	
008817FD	8D85 30FFFFFF	lea eax,dword ptr ss:[ebp-D0]	
00881803	50	push eax	
00881804	8D85 24FFFFFF	lea eax,dword ptr ss:[ebp-DC]	
0088180A	50	push eax	
0088180B	53	push ebx	
0088180C	68 40668900	push shamoon.896640	896640: L"PROCESSOR_ARCHITECTURE"
00881811	FFB5 28FFFFFF	push dword ptr ss:[ebp-D8]	
00881817	89D0 24FFFFFF	mov dword ptr ss:[ebp-DC],ebx	
0088181D	C785 2CFFFFFF 640000	mov dword ptr ss:[ebp-D4],64	64: 'd'
00881827	FF15 0C608900	call dword ptr ds:[<&RegQueryValueEx>]	
0088182D	85C0	test eax,eax	

Summary

Before I started this project, I used a Windows 10 Pro virtual machine and isolated it from the network. While it comes to reverse engineering, it is better to isolate the device you are using to prevent the malware from taking action on your machine. Additionally, since we are using malware and running it, I took snapshots so that I do not lose my progress in case of emergency.

At the end of this project, I have more confidence using the tools we studied. Additionally, I have gained more knowledge and understanding of the process of malware engineering. I started my project with basic static analysis, which helped me understand what the malware might. Secondly, I started the basic dynamic analysis to see what malware changes would do to a victim machine. Generally, basic static analysis and basic dynamic analysis will help us understand and examine host-based indicators, network-based indicators, changed registries, and add services. These are important to us since we can create a checklist of items to go over in the advanced analysis. Thirdly, I used advanced static analysis, where I used IDA PRO to go over the malware code in assembly. Going through the malware itself and its functions and seeing the malware's actual implementation will let you understand the malware in depth. The importance of advanced static analysis is that we can examine if the code will accept any command-line argument or if any useful comments might change the behavior of the malware. Also, it can be useful since we can use it to look for DLLs calls and other Windows APIs function calls. Finally, I used advanced dynamic analysis; in this section, I used x32.exe debug, which helped me understand the malware workflow. Understanding the malware workflow will give you a detailed understanding of how to detect the malware easily or how to stop it.

Resources

Shamoon Malware Sample:

<https://github.com/ytisf/theZoo/tree/master/malwares/Binaries/Shamoon>

MSDN Documentation:

<https://docs.microsoft.com/en-us/>

Shamoon Malware Background:

<https://www.cleverfiles.com/help/shamoon-malware-analysis.html>

Shamoon Malware Functionality Explained:

<https://www.darkreading.com/attacks-breaches/shades-of-shamoon-new-disk-wiping-malware-targets-middle-east-orgs/d/d-id/1336520>