Rochester Institute of Technology

2021

Rochester Institute of Technology,
B. Thomas Golisano College of Computing and Information Sciences

Shamoon v.2012

REVERSE ENGINEERING PROJECT FOR CSEC.202.03

ABDULMALIK K. BANASER

Rochester Institute of Technology

Contents

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	• 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 projected 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.

Rochester Institute of Technology

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

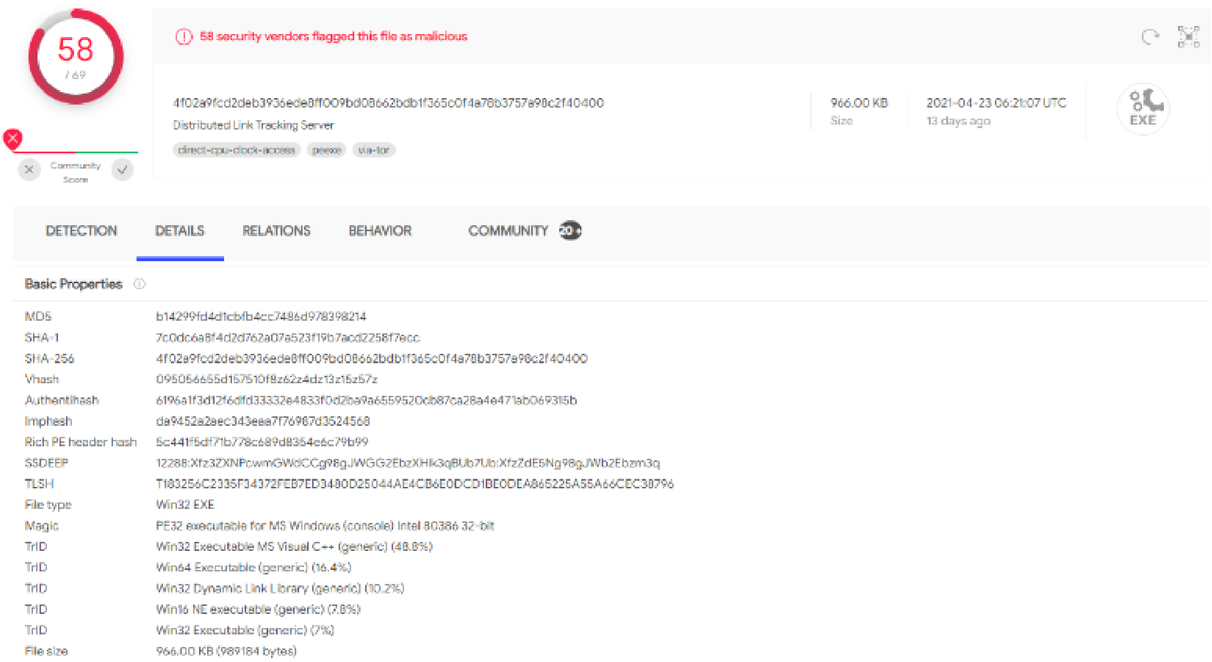
In this section, I used basic static tools that helped me to analyze the malware sample without running it. This is an important part of malware analysis process which can show many host-based indicators, network-based indicators, and other valuable information about the malware. After this step, I was 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 Shamoon.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\Shamoon.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.



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 .\Shamooon.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 .\Shamooon.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 .\Shamooon.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 .\Shamooon.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 .\Shamooon.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.

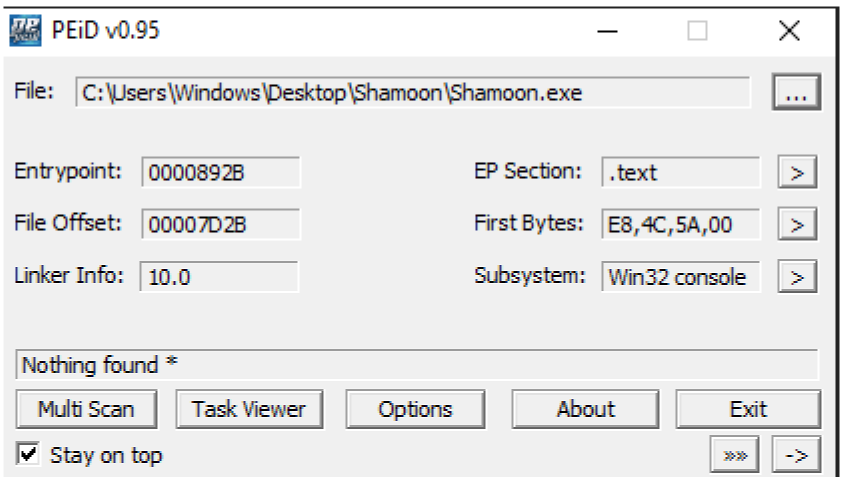
```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamooon.exe | findstr ".dll"
kernel32.dll
\system32\kernel32.dll
netapi32.dll
mscoree.dll
NETAPI32.dll
WS2_32.dll
KERNEL32.dll
USER32.dll
ADVAPI32.dll
SHELL32.dll
```

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.

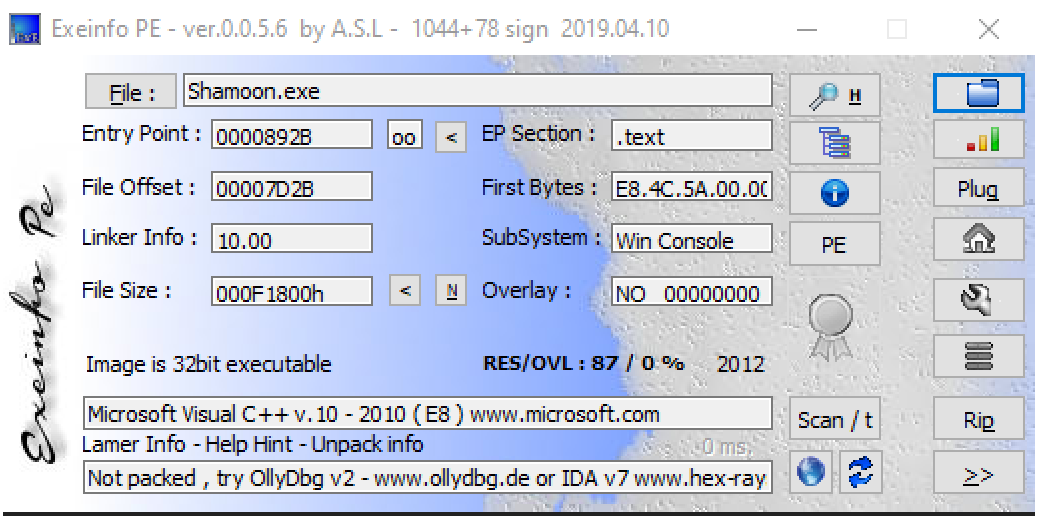
```
PS C:\Users\Windows\Desktop\Shamoon> strings .\Shamooon.exe | findstr "Services"
SYSTEM\CurrentControlSet\Services\TrkSvr
PS C:\Users\Windows\Desktop\Shamoon>
```

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.

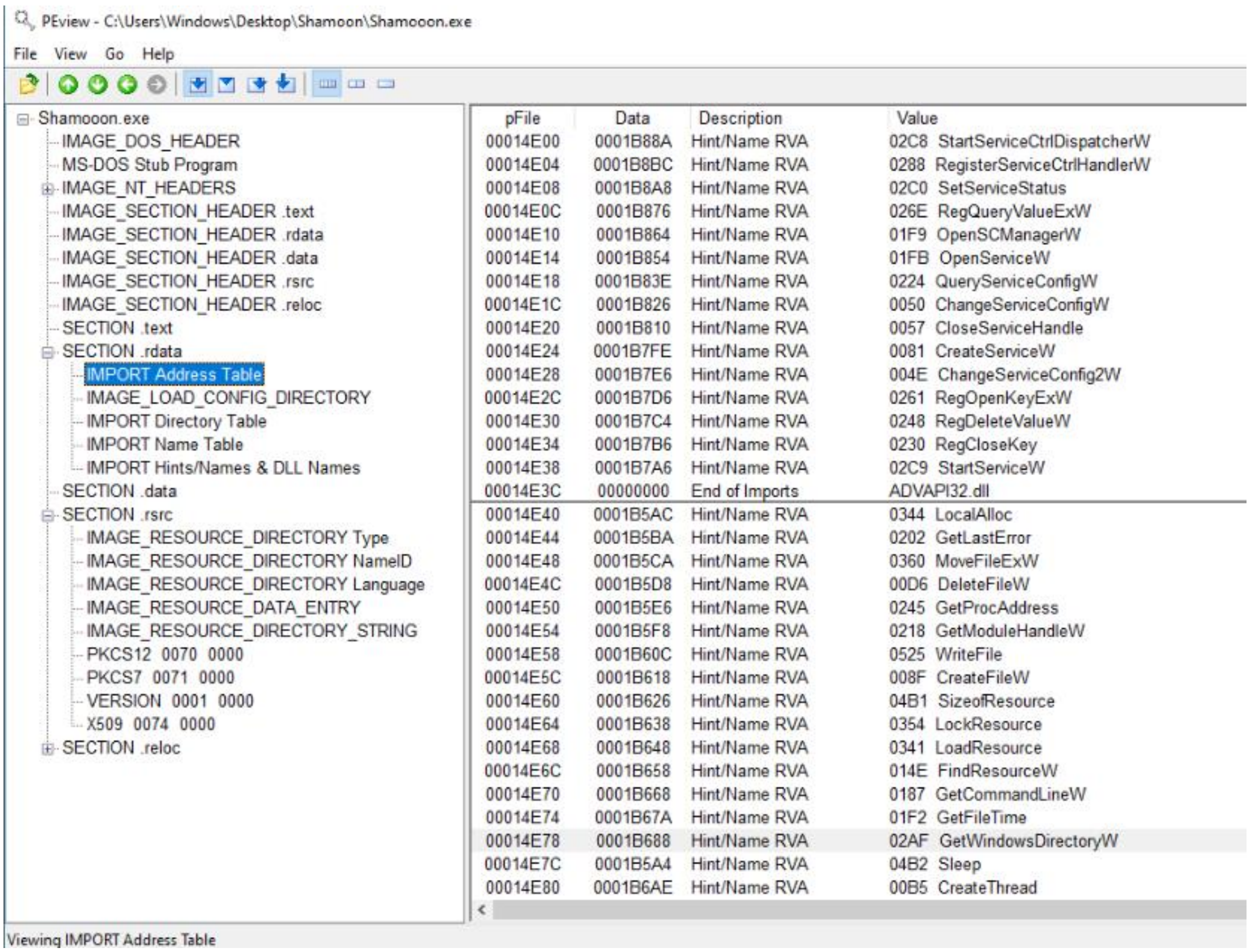
Third, I tried to examine whether the malware is packed or not, and check other information such as creation time, time stamps, DLL calls. For this purpose, I used "PEiD, Exeinfo PE, and PEview" tools.



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.

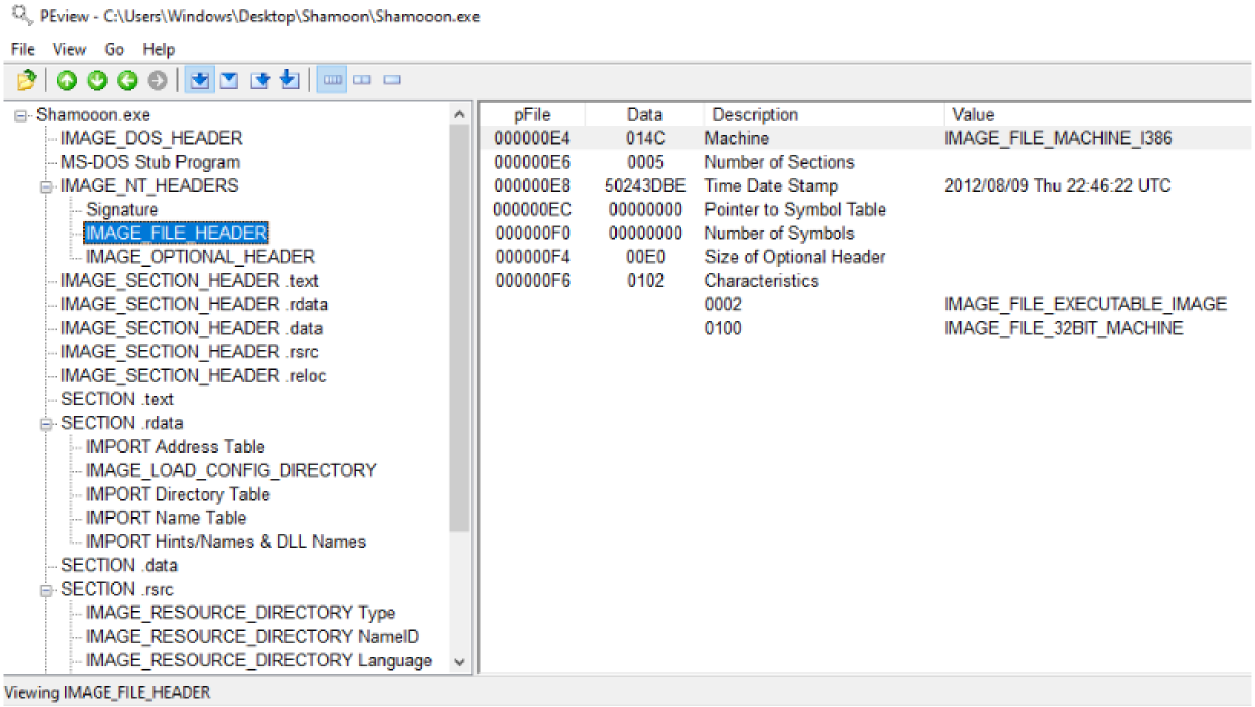


Then, I moved to check the DLL calls and imported functions to gain better understanding of the malware.

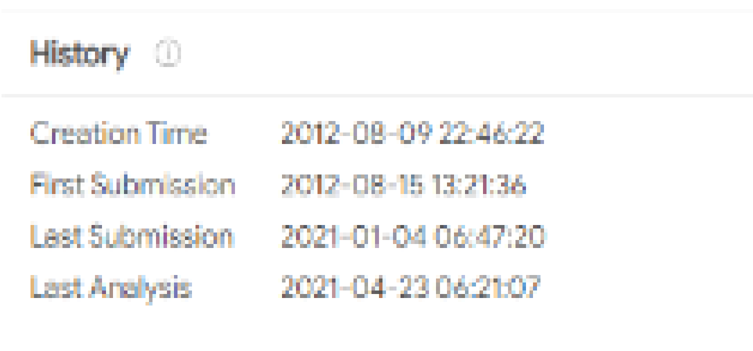


pFile	Data	Description	Value
00014E00	0001B88A	Hint/Name RVA	02C8 StartServiceCtrlDispatcherW
00014E04	0001B8BC	Hint/Name RVA	0288 RegisterServiceCtrlHandlerW
00014E08	0001B8A8	Hint/Name RVA	02C0 SetServiceStatus
00014E0C	0001B876	Hint/Name RVA	026E RegQueryValueExW
00014E10	0001B864	Hint/Name RVA	01F9 OpenSCManagerW
00014E14	0001B854	Hint/Name RVA	01FB OpenServiceW
00014E18	0001B83E	Hint/Name RVA	0224 QueryServiceConfigW
00014E1C	0001B826	Hint/Name RVA	0050 ChangeServiceConfigW
00014E20	0001B810	Hint/Name RVA	0057 CloseServiceHandle
00014E24	0001B7FE	Hint/Name RVA	0081 CreateServiceW
00014E28	0001B7E6	Hint/Name RVA	004E ChangeServiceConfig2W
00014E2C	0001B7D6	Hint/Name RVA	0261 RegOpenKeyExW
00014E30	0001B7C4	Hint/Name RVA	0248 RegDeleteValueW
00014E34	0001B7B6	Hint/Name RVA	0230 RegCloseKey
00014E38	0001B7A6	Hint/Name RVA	02C9 StartServiceW
00014E3C	00000000	End of Imports	ADVAPI32.dll
00014E40	0001B5AC	Hint/Name RVA	0344 LocalAlloc
00014E44	0001B5BA	Hint/Name RVA	0202 GetLastError
00014E48	0001B5CA	Hint/Name RVA	0360 MoveFileExW
00014E4C	0001B5D8	Hint/Name RVA	00D6 DeleteFileW
00014E50	0001B5E6	Hint/Name RVA	0245 GetProcAddress
00014E54	0001B5F8	Hint/Name RVA	0218 GetModuleHandleW
00014E58	0001B60C	Hint/Name RVA	0525 WriteFile
00014E5C	0001B618	Hint/Name RVA	008F CreateFileW
00014E60	0001B626	Hint/Name RVA	04B1 SizeofResource
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

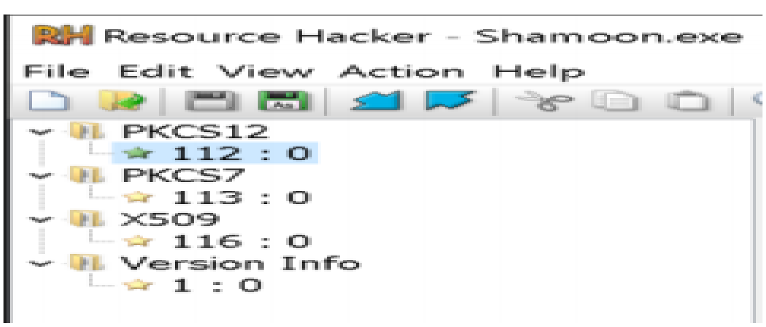
After examining the "IMPORT Address Table," I was able to get an overview of the functionality of the malware, as I noticed many function calls that are used to manipulate the file system, process, services, and registries.



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."



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.

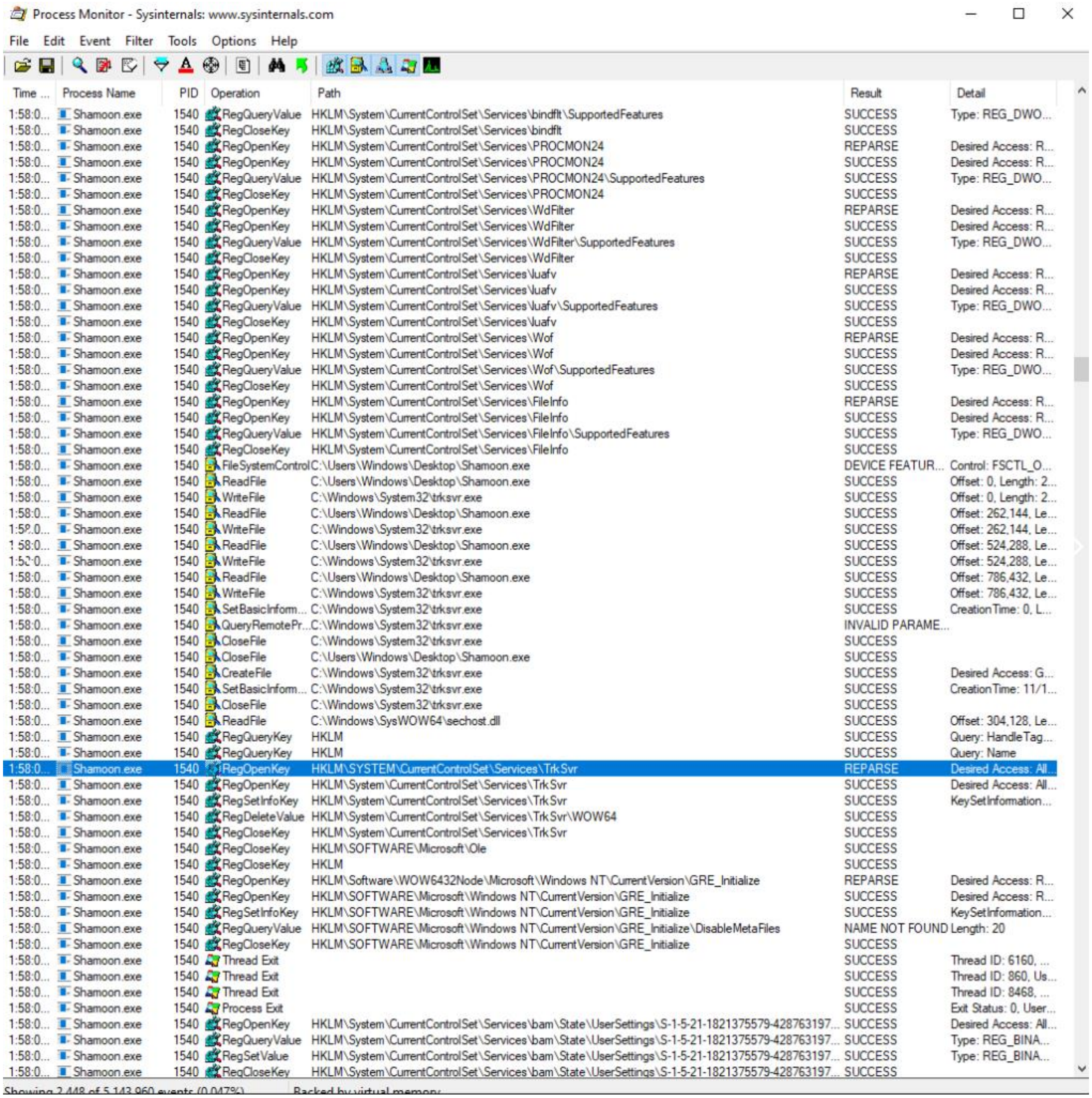
Analysis

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.

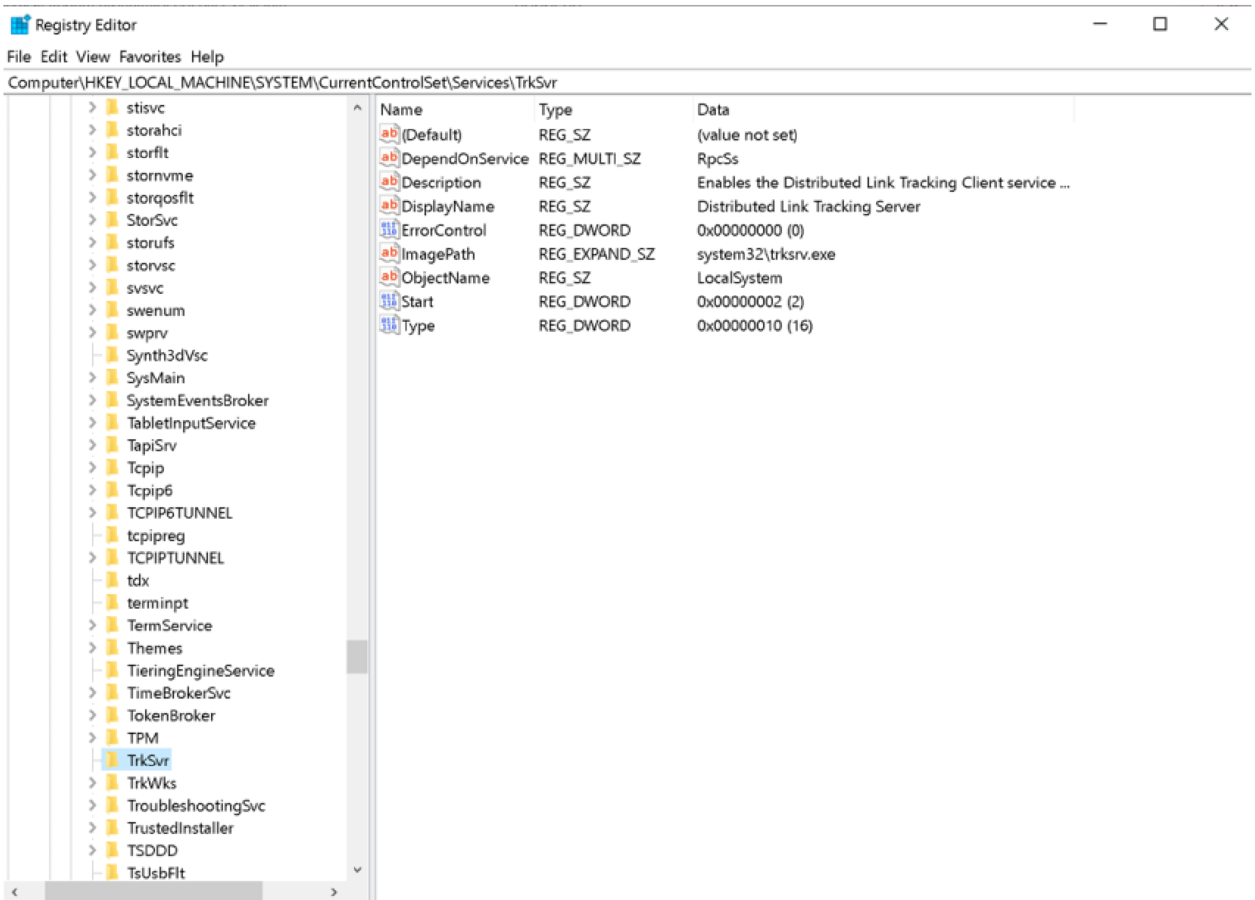
```
----------------------------------
Keys added: 35142
----------------------------------
HKLM\SYSTEM
HKLM\SYSTEM\ActivationBroker
HKLM\SYSTEM\ActivationBroker\Plugins
HKLM\SYSTEM\ActivationBroker\Plugins\{0913ACCF-B1A8-4EEE-A0C7-F4D7C12F4EEC}
HKLM\SYSTEM\ActivationBroker\Plugins\{14F3C12D-7712-42CC-B7CC-64D288560C43}
HKLM\SYSTEM\ActivationBroker\Plugins\{17B21A1B-6C59-48E0-A448-6BC9AD2C58FE}
HKLM\SYSTEM\ActivationBroker\Plugins\{5672B88A-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-ADF78D8ED45A}
HKLM\SYSTEM\ActivationBroker\Plugins\{AC59432D-8659-48C4-A584-AFEBC920256F}
HKLM\SYSTEM\ActivationBroker\Plugins\{C2745EC3-CF23-4601-92EF-D189B711F933}
HKLM\SYSTEM\ActivationBroker\Plugins\{D6AC71F0-D4A7-41DD-88C4-B998585SD546}
HKLM\SYSTEM\ActivationBroker\Plugins\{F00006F2-44BC-44EF-808B-B26002A183C2}
HKLM\SYSTEM\ActivationBroker\Plugins\{F22D2A32-F1F4-4D62-AF5E-E5E8253AC6A6}
HKLM\SYSTEM\ActivationBroker\Plugins\{F48B770A-CBE5-44C2-8D4F-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
----------------------------------
HKLM\SOFTWARE\Google\Update\LastStartedAU: 0x5EA94C05
HKLM\SOFTWARE\Google\Update\LastStartedAU: 0x5EA9552F
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\goopdate_main:  67 01 00 00 00 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\goopdate_main:  69 01 00 00 00 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\goopdate_constructor:  67 01 00 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\goopdate_constructor:  69 01 00 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\cup_ecdsa_total:  E9 04 00 00 00 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\cup_ecdsa_total:  F1 04 00 00 00 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\worker_update_check_total:  9E 00 00 00 0
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\worker_update_check_total:  9F 00 00 00 0
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\cup_ecdsa_http_failure:  E8 04 00 00 00 0
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Counts\cup_ecdsa_http_failure:  F0 04 00 00 00 0
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Integers\last_started_au:  05 4C A9 5E 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Integers\last_started_au:  2F 55 A9 5E 00 00 00
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Timings\updatecheck_failed_ms:  9D 00 00 00 00 0
HKLM\SOFTWARE\Google\Update\UsageStats\Daily\Timings\updatecheck_failed_ms:  9E 00 00 00 00 0
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows\CurrentVersion\E
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows\CurrentVersion\E
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows\CurrentVersion\S
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows\CurrentVersion\S
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows NT\CurrentVersio
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows NT\CurrentVersio
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows NT\CurrentVersio
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Microsoft\Windows NT\CurrentVersio
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Classes\Local Settings\Software\Mi
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Classes\Local Settings\Software\Mi
HKU\S-1-5-21-2112901208-2556333921-550770388-1000\Software\Classes\Local Settings\Software\Mi

HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W
HKU\S-1-5-21-2112901208-2556333921-550770388-1000_Classes\Local Settings\Software\Microsoft\W

----------------------------------
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.



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.

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.

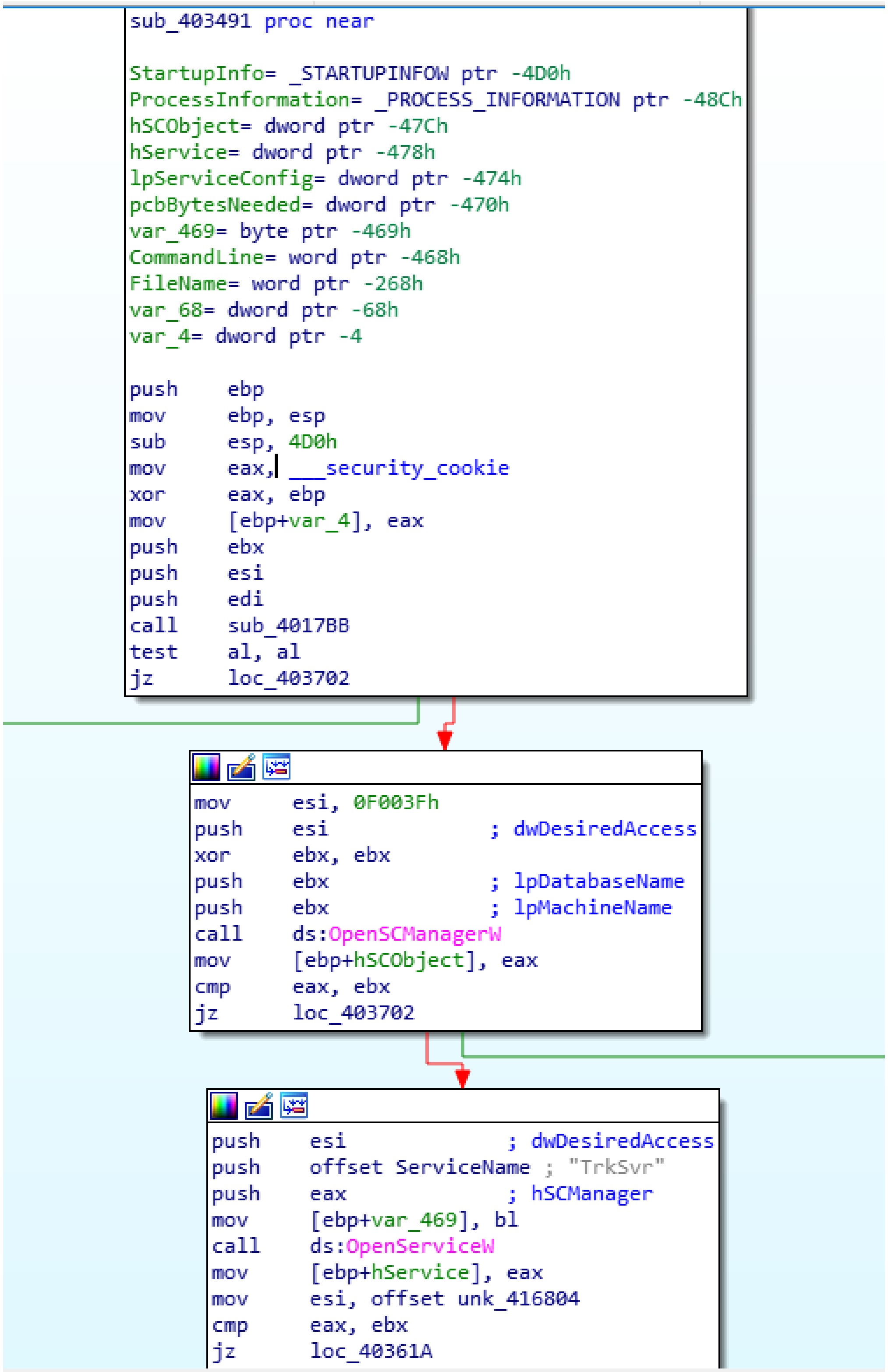
```
; Attributes: bp-based frame

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
```

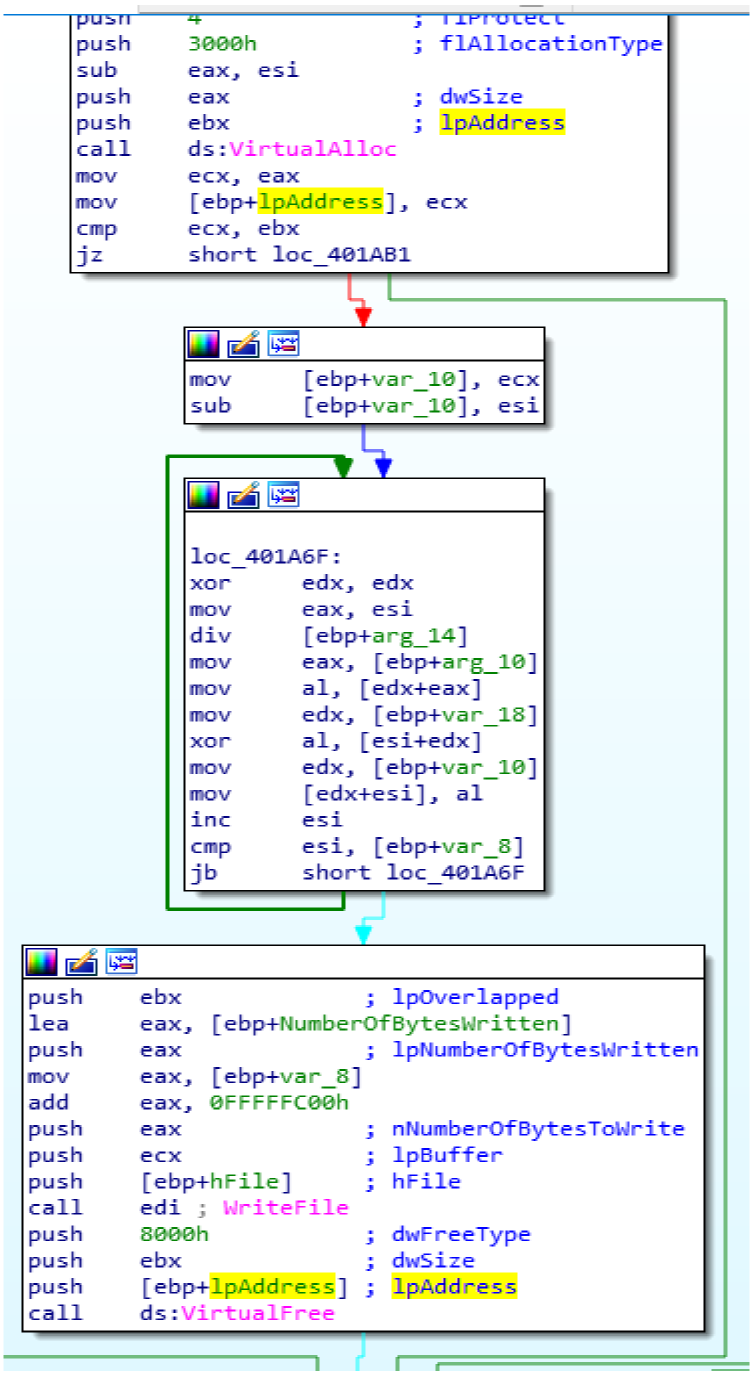
However, I continued to go over the malware where I noticed another behavior of the malware where it will attempt to create and run a new service called “TrkSvr,” as we saw before in the basic dynamic analysis.



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."

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.



After that I tired 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+136↑o
.data:0041C431                 db  7Fh ;
.data:0041C432                 db  5Dh ; ]
.data:0041C433                 db 0FBh ; û
```

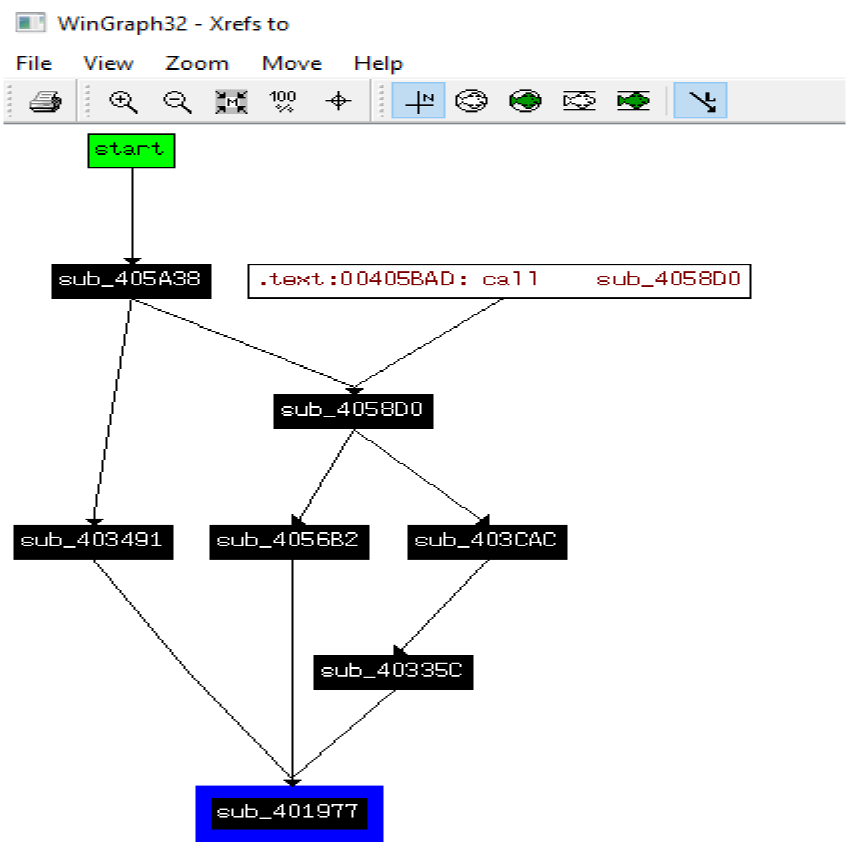
Which is called pushed as an argument inside all the resource's functions to other function that load and use the resources. Additionally, we can see that different resources will use different decryption keys

```
loc_4035B9:
call    nullsub_1
push    4                   ; int
push    offset unk_41C438 ; int
push    offset aX509        ; 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
```

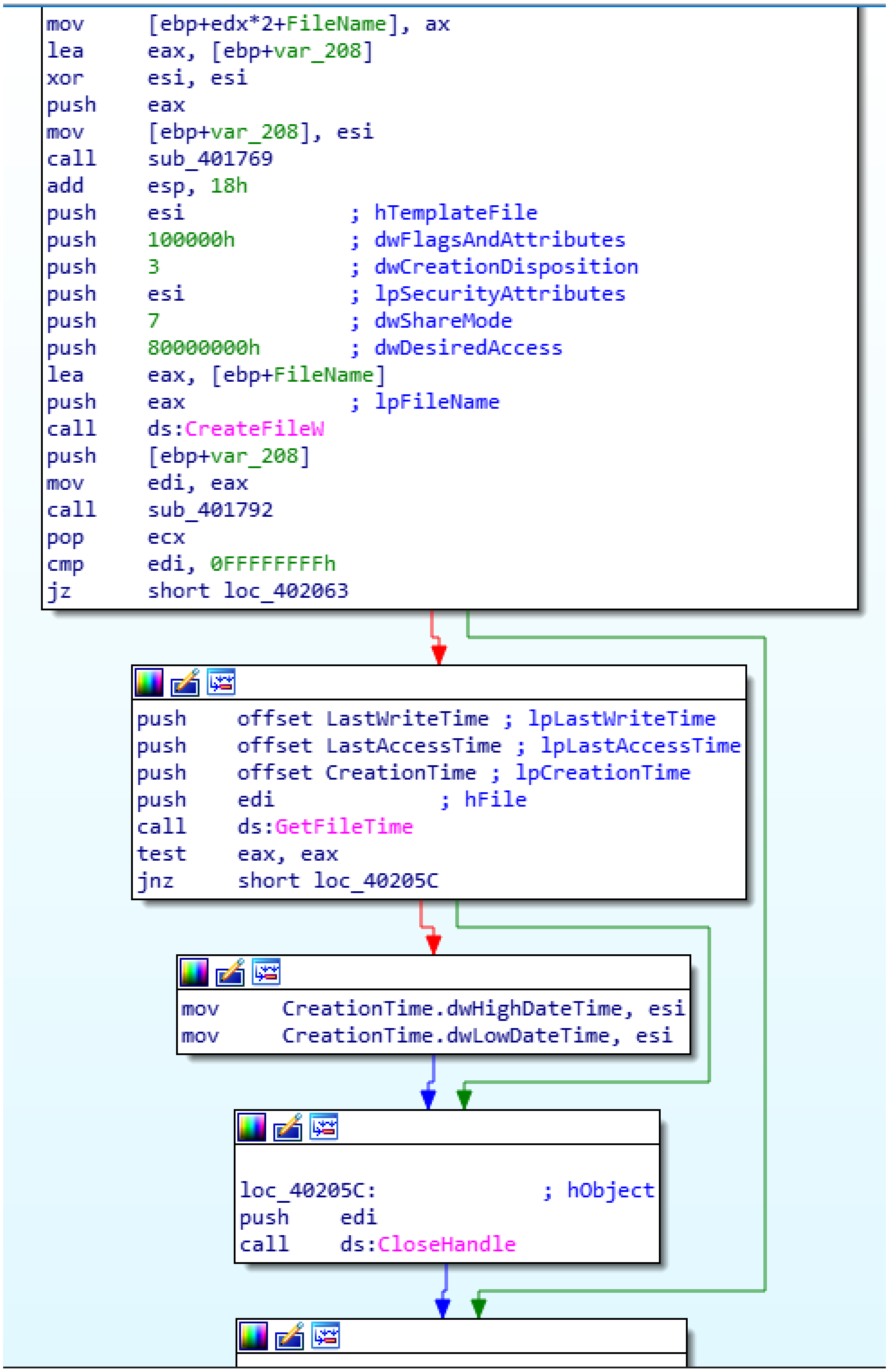
```
loc_4057E6:                 ; int
push    4
push    offset unk_41C430 ; int
push    offset aPkcs12    ; "PKCS12"
push    70h               ; lpName
lea     eax, [ebp+FileName]
push    ebx               ; hModule
push    eax               ; lpFileName
call    sub_401977
add     esp, 18h
test    al,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

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.



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 aKijjjjnsnjbnnc ; "kijjjjnsnjbnncbknbkjadc\r\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

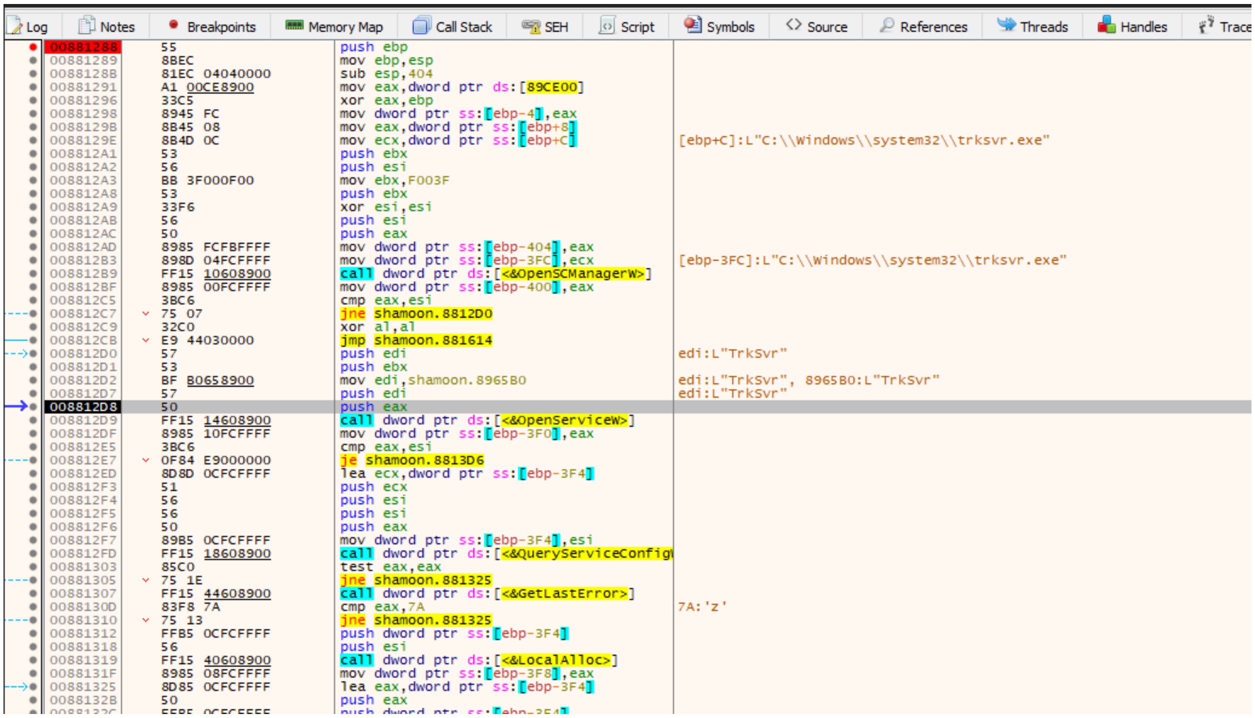
First, I checked the DLL imports to check if there are any runtime linking to the malware sample we are using. I found a similar output of what we found using basic static analysis.

Base	Module	Party	Path
00880000	shamoon.exe	User	C:\Users\Windows\Desktop\Shamoon.exe
6DAA0000	schedcli.dll	System	C:\Windows\SysWOW64\schedcli.dll
736F0000	srvcli.dll	System	C:\Windows\SysWOW64\srvcli.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	msvcp_win.dll	System	C:\Windows\SysWOW64\msvcp_win.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

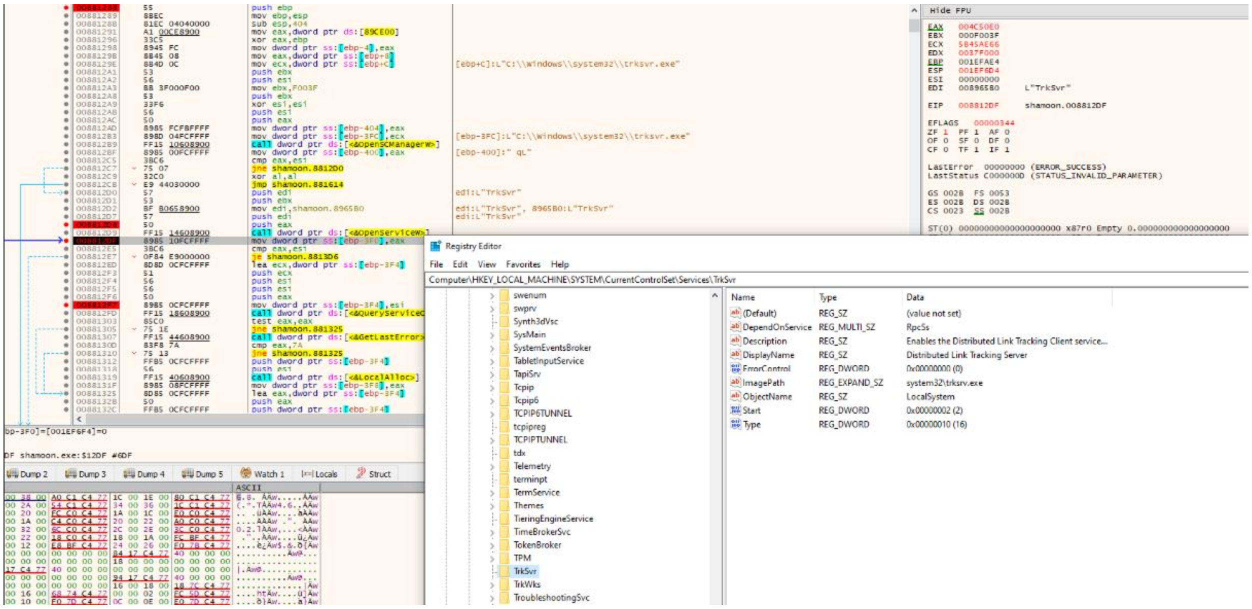
After checking the DLLs, I looked for the "EntryPoint," as it will lead us to the main function so we can see the true functionality of the malware.

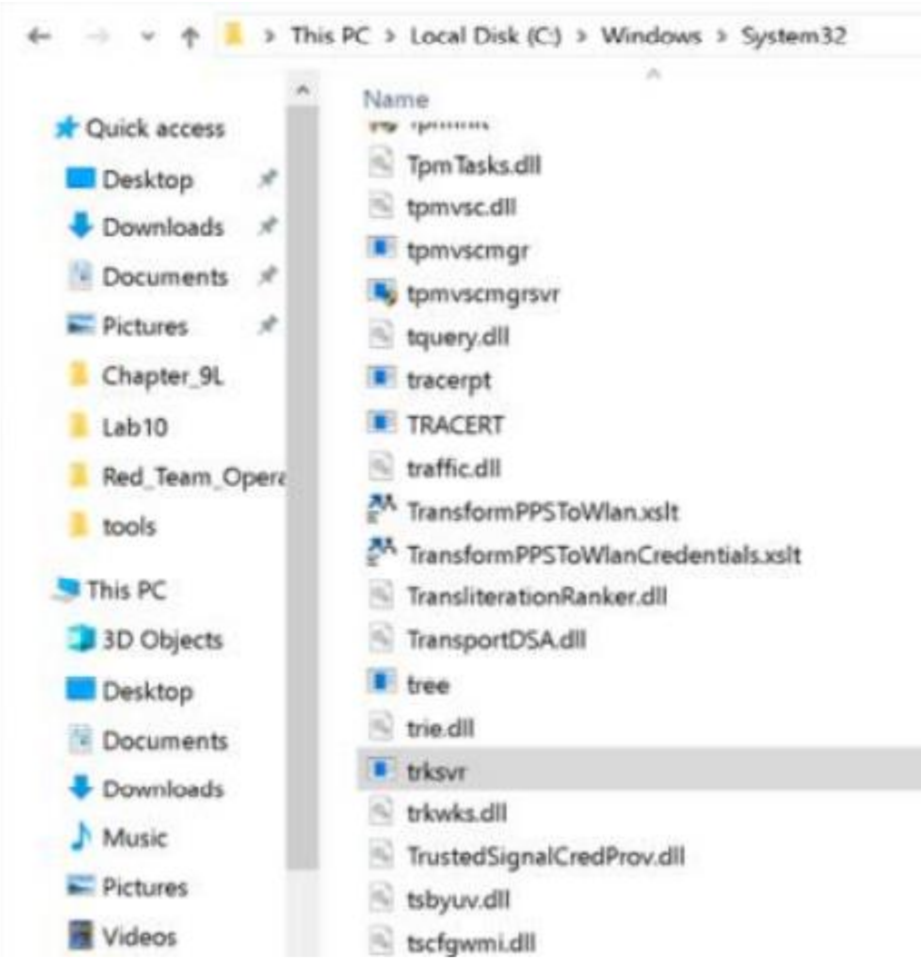
```
0088890A   837D E4 00          cmp dword ptr ss:[ebp-1C],0
0088890E   75 06               jne shamoon.888916
00888910   50                  push eax
00888911   E8 39FEFFFF         call shamoon.88874F
00888916   E8 59FEFFFF         call shamoon.888774
0088891B   C745 FC FEFFFFFF    mov dword ptr ss:[ebp-4],FFFFFFFE
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.

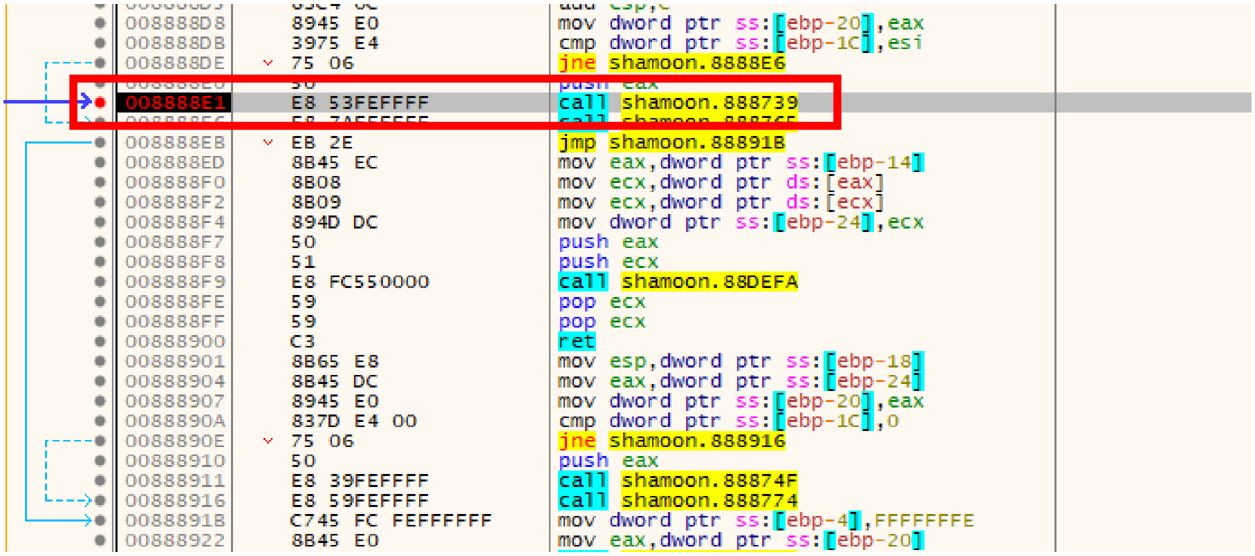


However, after step over few times, I was able to see the actual executable file, and the registry key added. This indicate that the malware is running as we expected.





Third, I patched the malware to see what changes will happen in its behavior. To do so, I have changed on of the if-statement instruction to return 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 wired behavior of the malware, but eventually it will crash. So, I guess there is nothing interesting in there.



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.

Also, I was able to see the functionality of the malware where it uses the same service trying to duplicate itself around the connected domain network, which will help it to spread faster.

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

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.

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