Roberto Reade

CSEC 202.03

Reverse Engineering Fundamentals

Professor Weissman

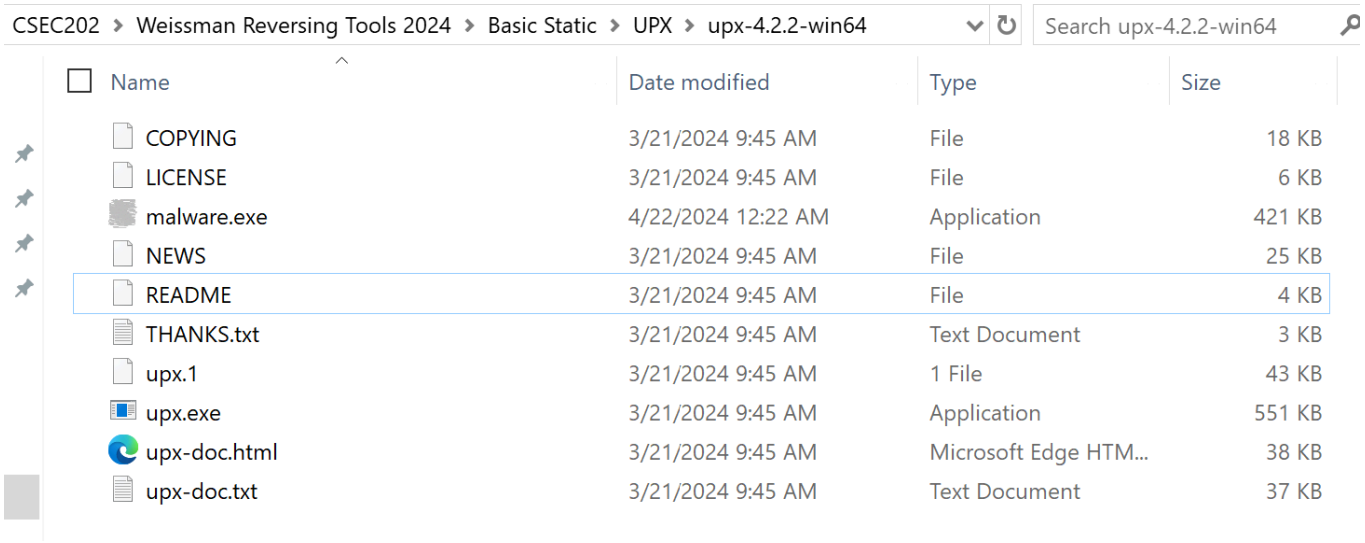
Reversing Project

INTRODUCTION:

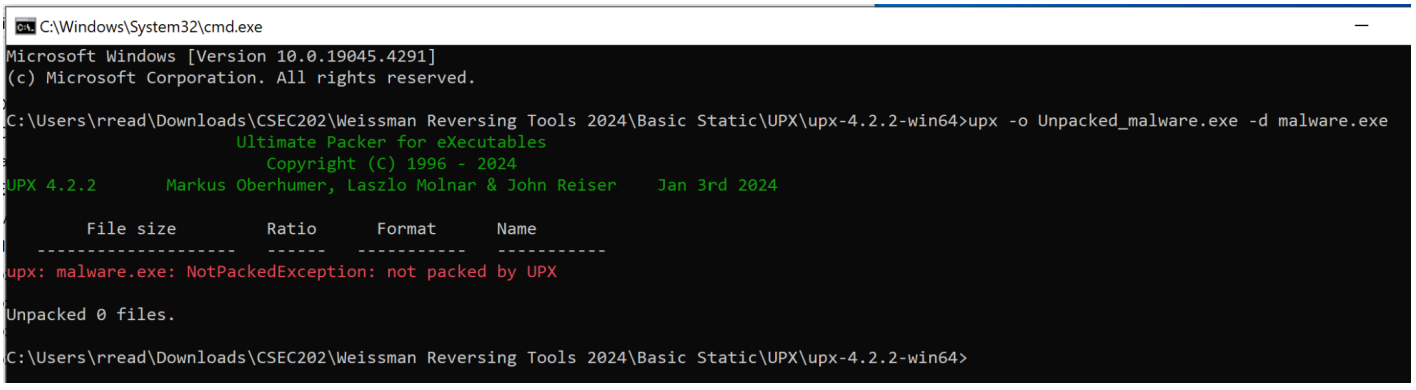
The malware I will be analyzing and testing for this project is Lumma Stealer. This piece of malware can steal sensitive information from a user's computer. More specifically, it can target data on a user's system, browsers, and browser extensions.

Basic Static Analysis:

UPX:

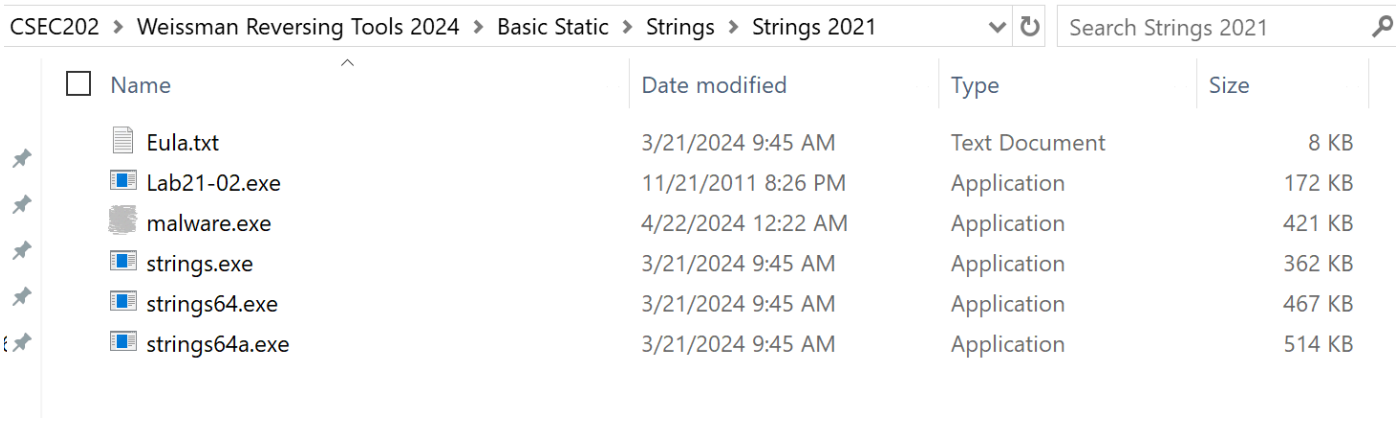


The first Basic static tool I used was UPX. I am using UPX to check if the Lumma Stealer is packed by UPX. In order to do this, I first moved the malware into the UPX folder shown in the screenshot above. This would then allow me to open the command prompt from file explorer and have access to the malware.

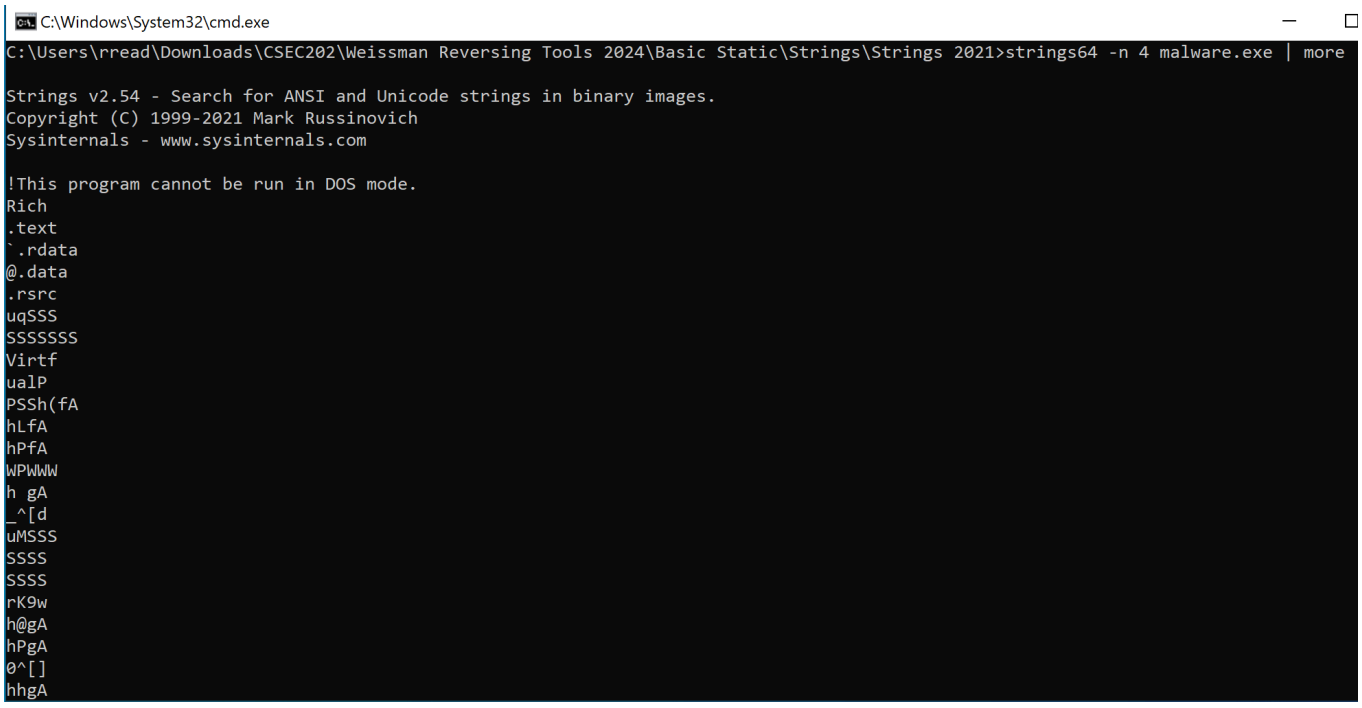


In the screenshot above, I entered the command:
“upx -o Unpacked_malware.exe -d malware.exe”
This command decompresses, and unpacks the malware.exe file. It then writes it to an output file. In this case I made the output file “Unpacked_malware.exe”. Once this command was run, I noticed that the malware I am using is not packed by UPX. This is seen in the screenshot above in the red lettering.

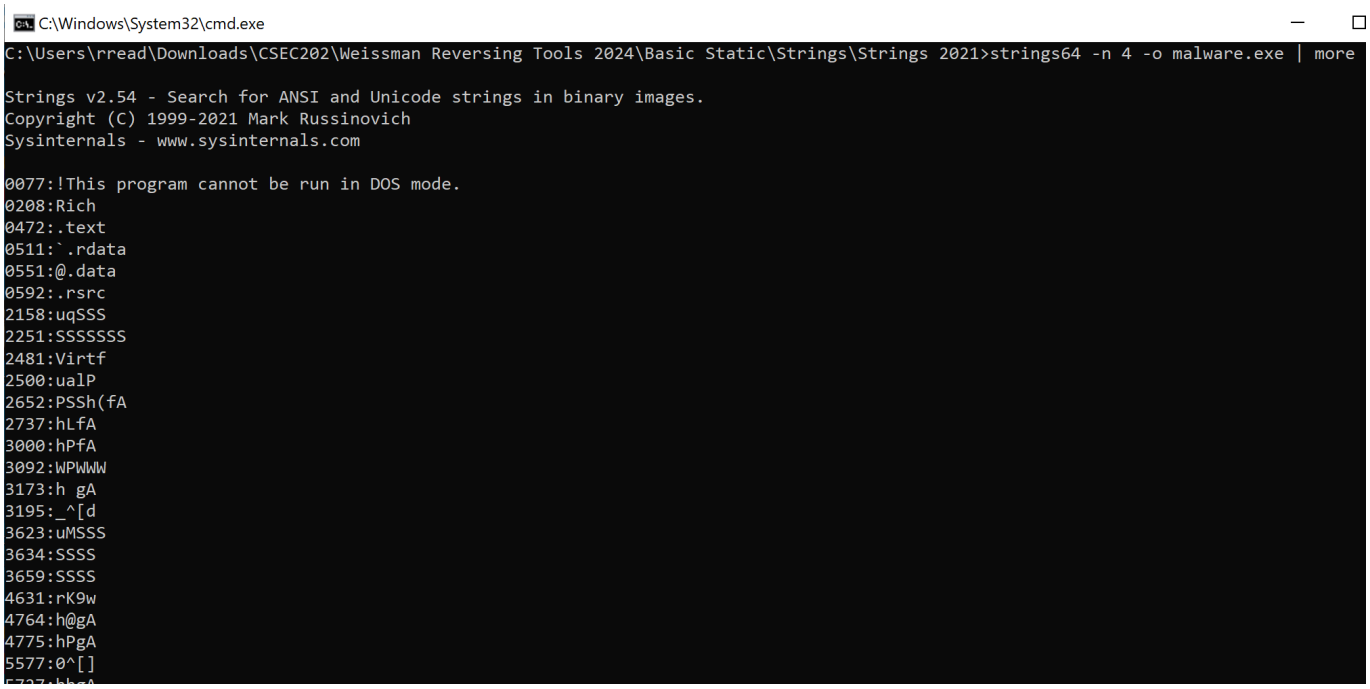
Strings:



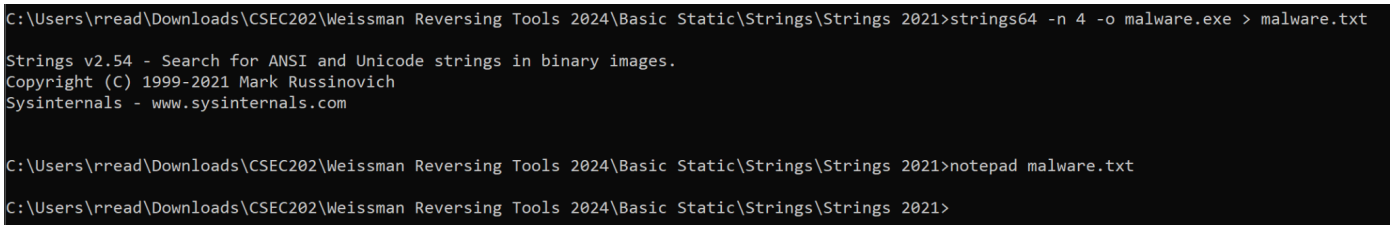
The next tool I used was Strings. This tool will look for strings of ASCII values in the malware.exe file and print them out for me to see. In order to use Strings, I first had to move the malware.exe file into the Strings folder as shown in the above screenshot. This allowed me to open the command prompt from the file explorer and have access to the malware file.

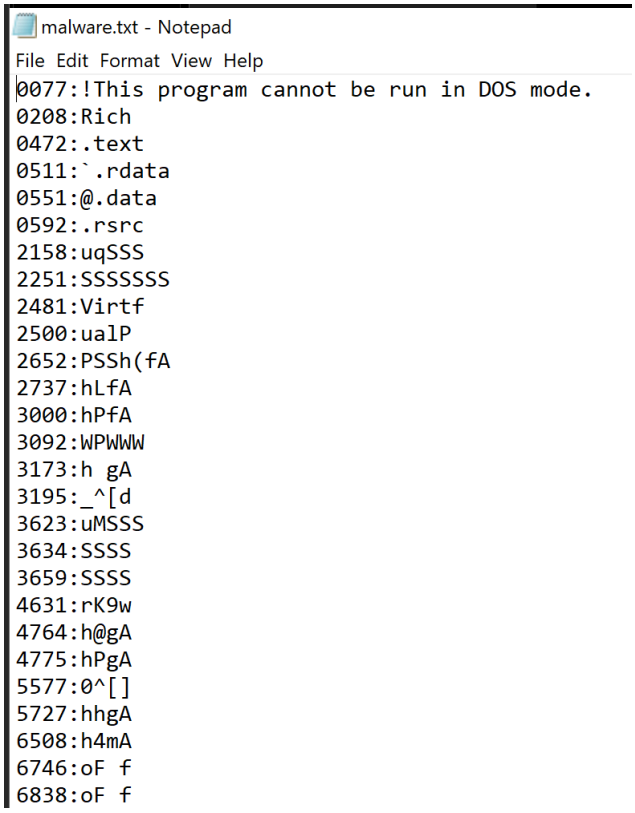


In the screenshot above, I used the command: “strings64 -n 4 malware.exe | more”. This command prints out all the strings in the malware.exe file that are a length of 4 or above. I did this since a lot of strings below a length of 4 are for the most part garbage.



In the above screenshot, I basically used the same command. However, in this command, I added “-o”. The new command I entered was: “strings64 -n 4 -o malware.exe | more”. This command and the “-o” allowed me to see the offset of the strings that were printed out. This helped me figure out which strings were used together if any.





The last command I entered in Strings was: “strings64 -n 4 -o malware.exe > malware.txt”. This command took the output of the Strings command and made it into a .txt file. I then used the command: “notepad malware.txt” in order to open the .txt file in Notepad. I used these commands in order to save the output of Strings for this file so I could go back and look at it later for reference while using other tools.

After going through the output of Strings on my “malware.exe” file, I noticed a lot of interesting things.

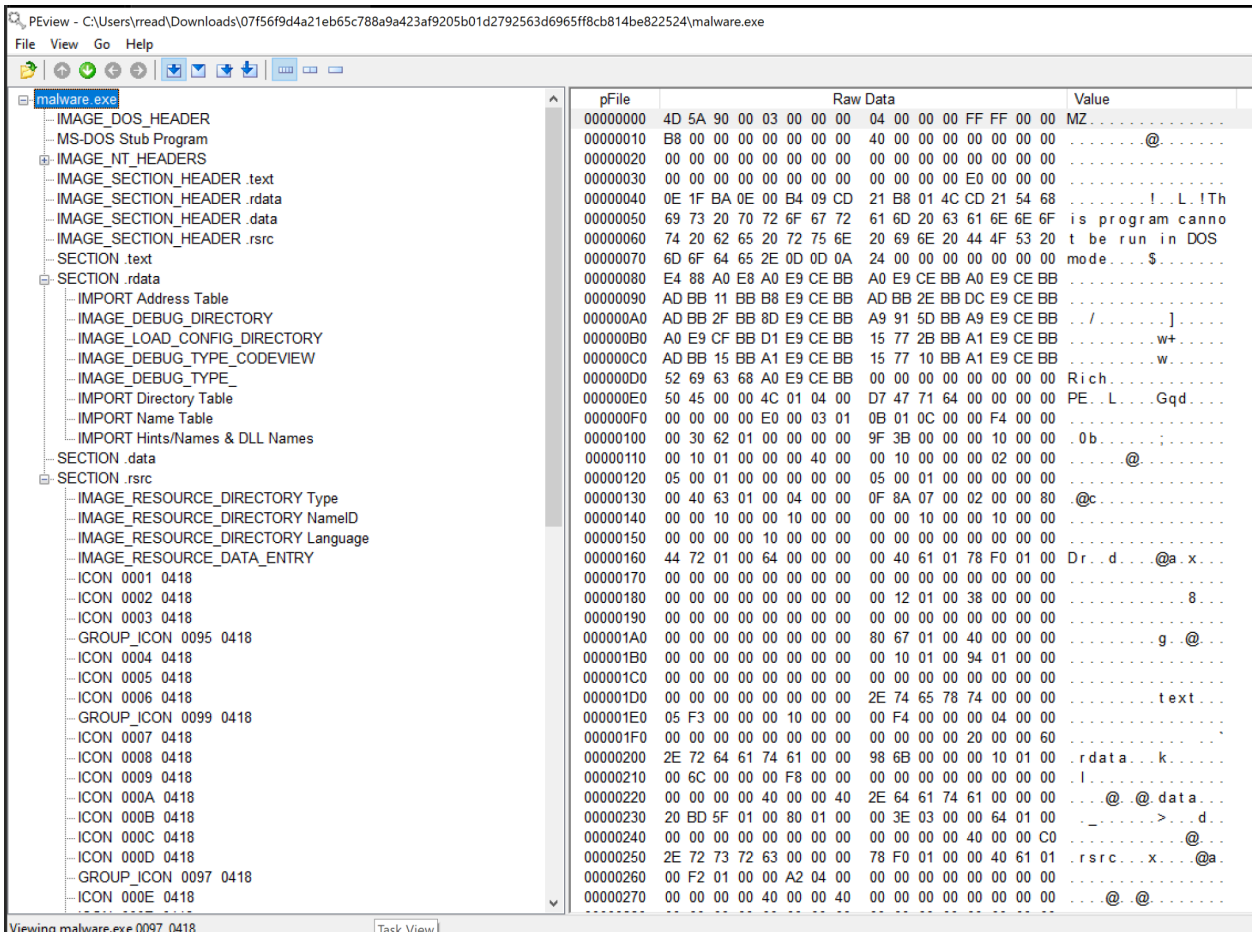
I noticed that there were error messages with corresponding error numbers, as well as notification messages. These messages could either be for the threat actor, or they could show up on the user's computer when the user attempts to do something. If these messages are for the threat actor, they would most likely pop up on their computer and let them know if the malware ran into an error, or notify them that something worked correctly and they now have access to the user's information.

I also noticed strings of the days of the week and months of the year. These strings can be used for multiple things. These strings could potentially be used to manipulate timestamps, or schedule certain tasks.These strings could also be used to get the date when data was collected or exfiltrated, manipulate timestamps to evade security analysts, and possibly display dates in different formats depending on the users location.

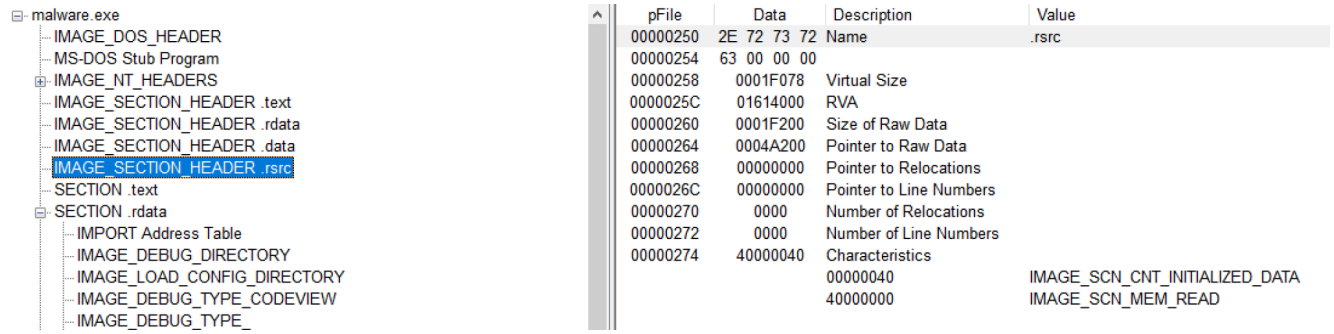
There were many getter, setter, create and delete functions that I noticed when looking through the strings. These getter functions can be used to get information needed for the malware to run, or get personal information from the user's device. The setter functions can be used to activate different parts of malware, or set values to certain files or other important information.

The delete files can be used to delete important data or information from the user's computer. Finally, the create functions can be used to create files, create processes, create registry entries, create threads, or even create network connections.
In Lumma Stealer, I noticed a large amount of strings in the format of two letters separated by a dash and then two capital letters following that dash. Each of these strings represents a language code and country or region code. This allows the malware to be run in various countries with various languages.
Finally, at the end of the output, I noticed many long strings with the same letter, and strings with a bunch of random letters that looked like gibberish. These strings can potentially be used for buffer overflow attacks on the user's device.

PEview:

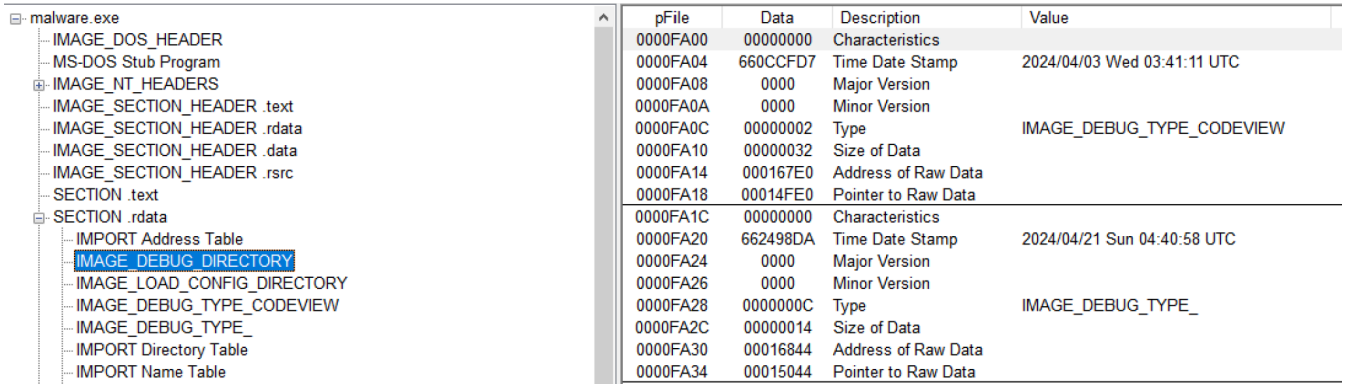


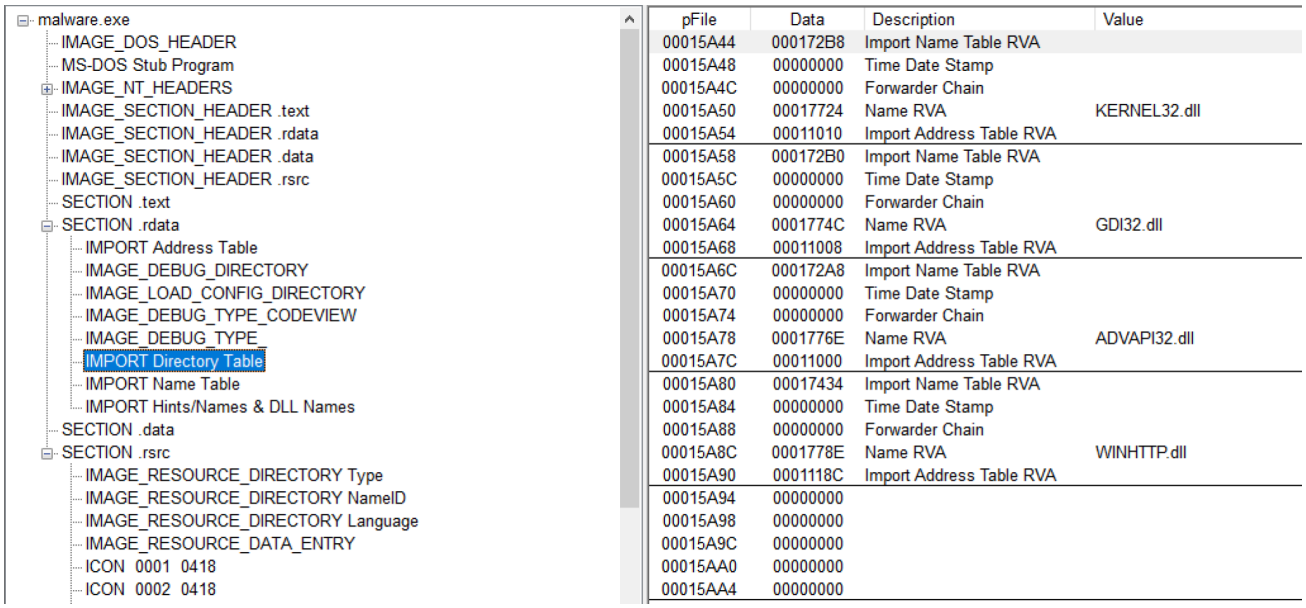
In order to view Lumma Stealer in PEview, I had to open the malware in PEview as shown in the above screenshot. Once it opened, I noticed some differences from the binaries we examined in class.
In the IMAGE_NT_HEADERS section, there are four IMAGE_SECTION_HEADERS. These include .text, .rdata, .data, and .rsrc. A screenshot of one of these IMAGE_SECTION_HEADERS is shown below.



As you can see in the screenshot above, there are sections for the pFile, Data, Description, and value. Within this IMAGE_SECTION_HEADER.rsrc, it gives information on the size of the data as well as pointer information.

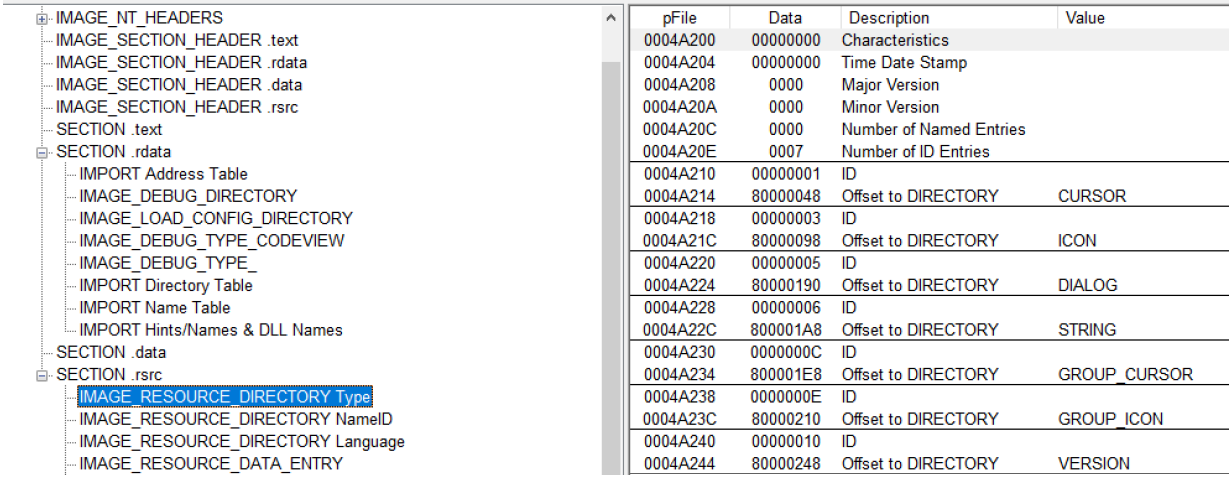
In the SECTION.rdata section, there are multiple IMPORT and IMAGE subsections. Below I will include a screenshot of an IMPORT and an IMAGE section.



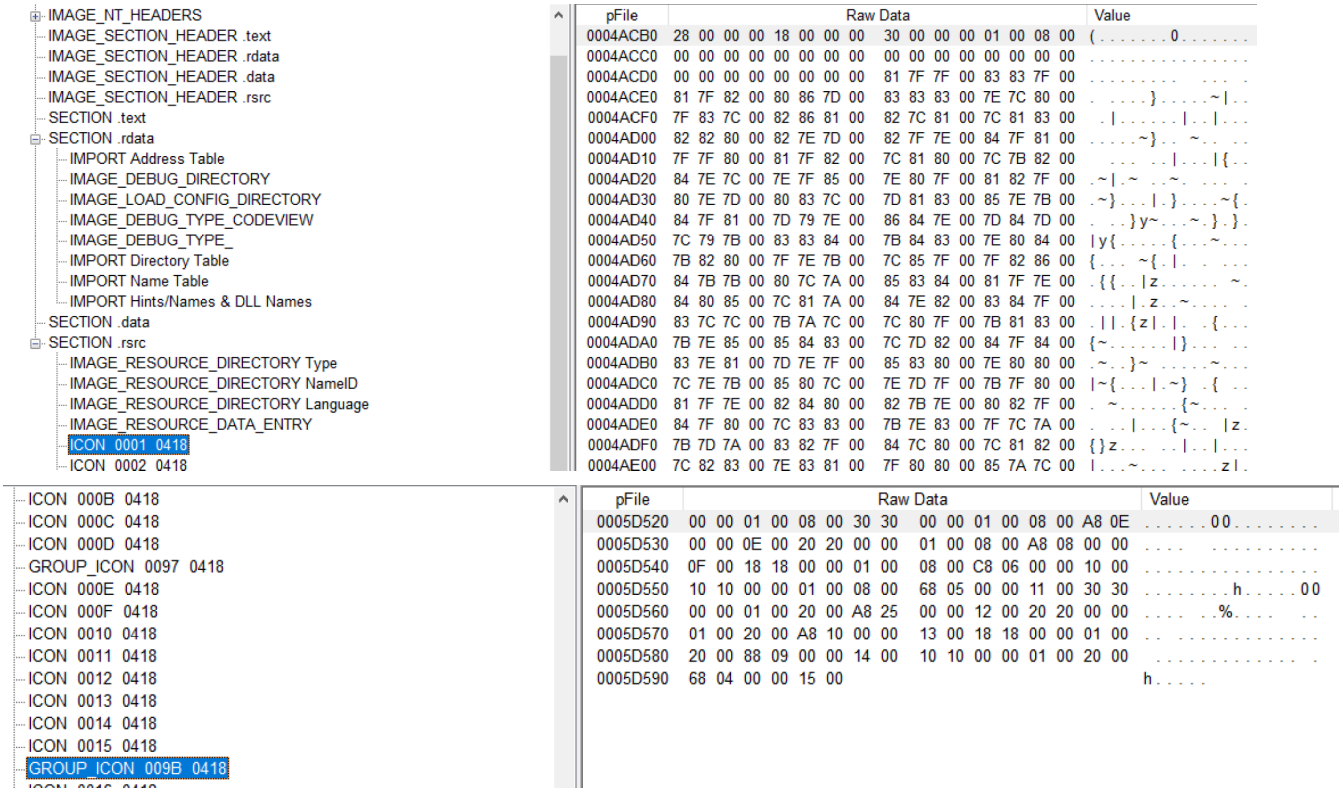


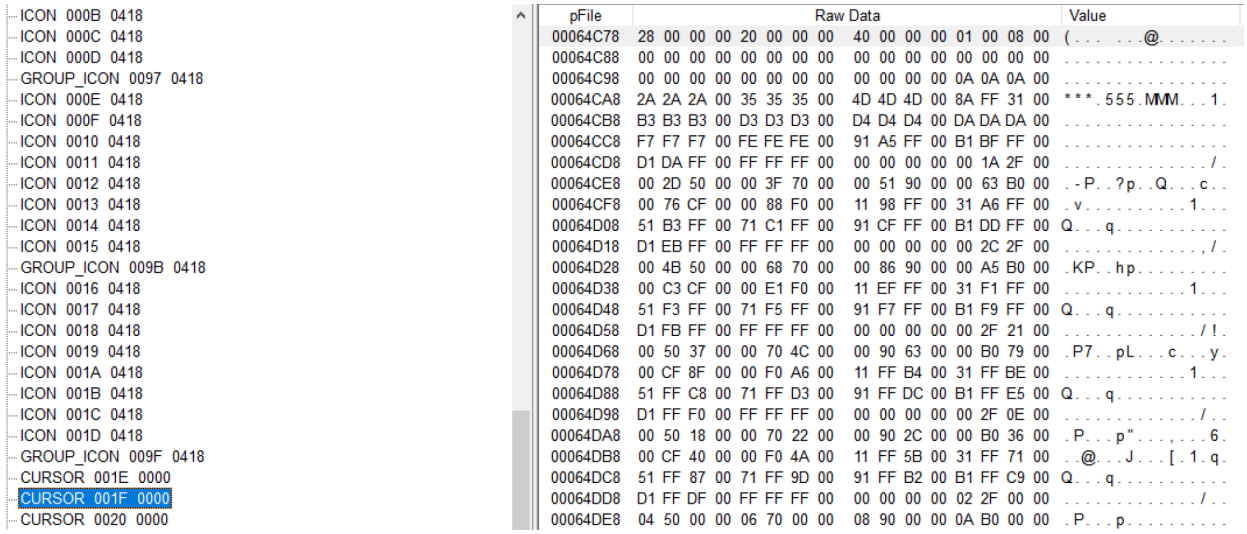
As you can see in the screenshots above, the IMAGE and IMPORT sections also give you sections for pFile, Data, Description, and Value. In each IMAGE section it gives information regarding the characteristics, timestamp, the versions, and information on the data. In each

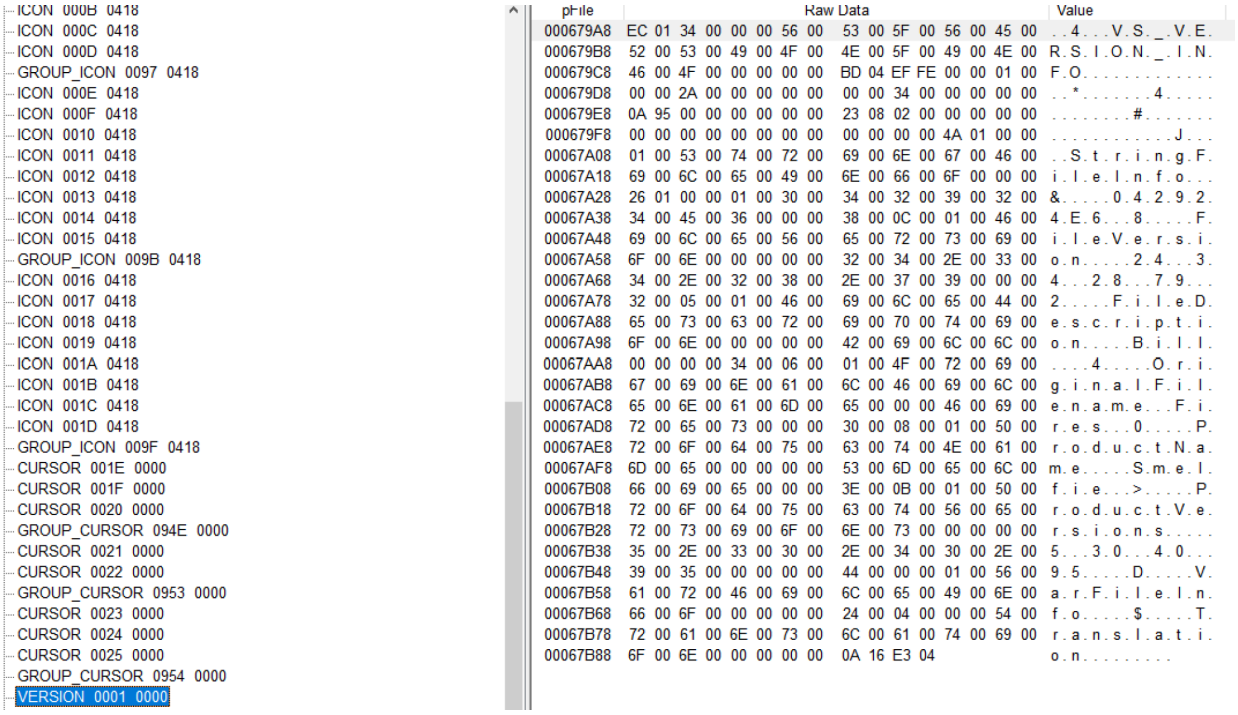
IMPORT section, it gives you information regarding the name RVA as well as other information regarding the imports and functions used.
In the SECTION.rsrc section, there are many subsections. These include IMAGE, ICON, GROUP, CURSOR, VERSION, and STRING. As you can tell, there is no binary in this section for me to extract like we did in labs and in class. This is because the malware I am using is not a binary. Below I will include a screenshot from each subsection in SECTION.rsrc.

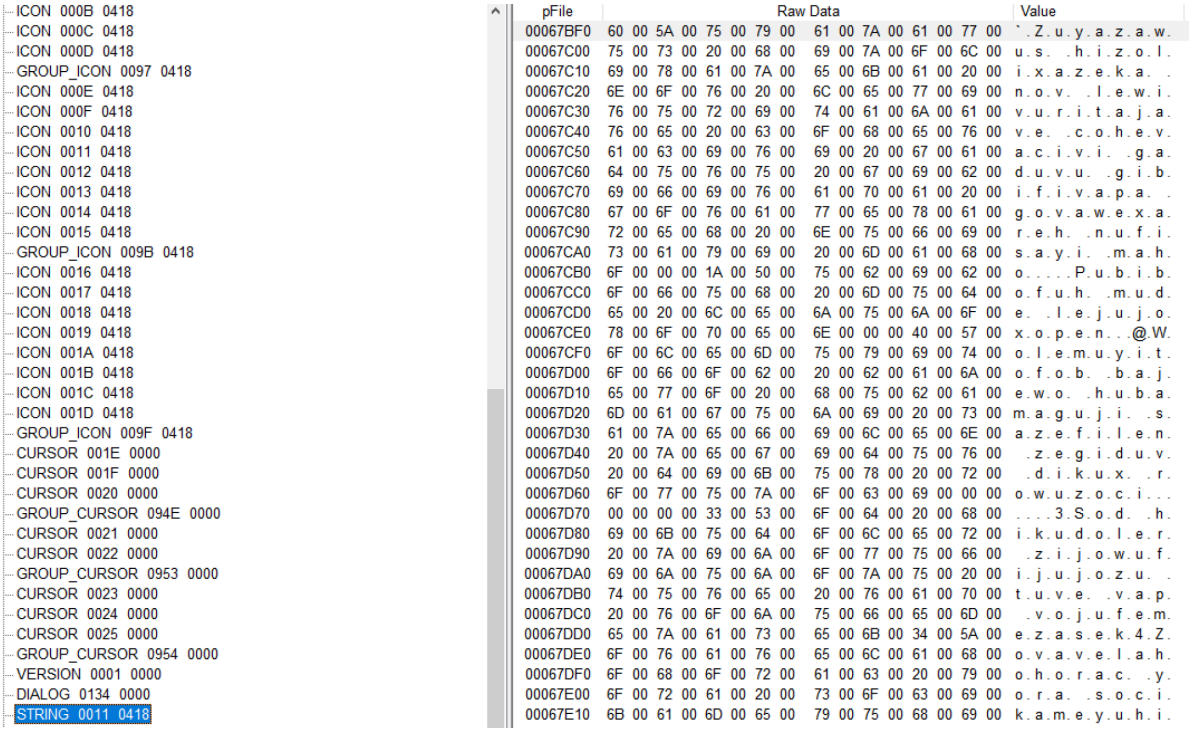


The IMAGE section, as seen previously, includes pFile, Data, Description, and Value. It gives information about the ID and Offset to DIRECTORY for every value.




ICON 000B 0418
ICON 000C 0418
ICON 000D 0418
GROUP_ICON 0097 0418
ICON 000E 0418
ICON 000F 0418
ICON 0010 0418
ICON 0011 0418
ICON 0012 0418
ICON 0013 0418
ICON 0014 0418
ICON 0015 0418
GROUP_ICON 009B 0418
ICON 0016 0418
ICON 0017 0418
ICON 0018 0418
ICON 0019 0418
ICON 001A 0418
ICON 001B 0418
ICON 001C 0418
ICON 001D 0418
GROUP_ICON 009F 0418
CURSOR 001E 0000
CURSOR 001F 0000
CURSOR 0020 0000
pFile Raw Data Value
00064C78 28 00 00 00 20 00 00 00 40 00 00 00 01 00 08 00
00064C88 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00064C98 00 00 00 00 00 00 00 00 00 00 00 00 0A 0A 0A 00
00064CA8 2A 2A 2A 00 35 35 35 00 4D 4D 4D 00 8A FF 31 00
00064CB8 B3 B3 B3 00 D3 D3 D3 00 D4 D4 D4 00 DA DA DA 00
00064CC8 F7 F7 F7 00 FE FE FE 00 91 A5 FF 00 B1 BF FF 00
00064CD8 D1 DA FF 00 FF FF FF 00 00 00 00 00 00 1A 2F 00
00064CE8 00 2D 50 00 00 3F 70 00 00 51 90 00 00 63 B0 00
00064CF8 00 76 CF 00 00 88 F0 00 11 98 FF 00 31 A6 FF 00
00064D08 51 B3 FF 00 71 C1 FF 00 91 CF FF 00 B1 DD FF 00
00064D18 D1 EB FF 00 FF FF FF 00 00 00 00 00 00 2C 2F 00
00064D28 00 4B 50 00 00 68 70 00 00 86 90 00 00 A5 B0 00
00064D38 00 C3 CF 00 00 E1 F0 00 11 EF FF 00 31 F1 FF 00
00064D48 51 F3 FF 00 71 F5 FF 00 91 F7 FF 00 B1 F9 FF 00
00064D58 D1 FB FF 00 FF FF FF 00 00 00 00 00 00 2F 21 00
00064D68 00 50 37 00 00 70 4C 00 00 90 63 00 00 B0 79 00
00064D78 00 CF 8F 00 00 F0 A6 00 11 FF B4 00 31 FF BE 00
00064D88 51 FF C8 00 71 FF D3 00 91 FF DC 00 B1 FF E5 00
00064D98 D1 FF F0 00 FF FF FF 00 00 00 00 00 00 2F 0E 00
00064DA8 00 50 18 00 00 70 22 00 00 90 2C 00 00 B0 36 00
00064DB8 00 CF 40 00 00 F0 4A 00 11 FF 5B 00 31 FF 71 00
00064DC8 51 FF 87 00 71 FF 9D 00 91 FF B2 00 B1 FF C9 00
00064DD8 D1 FF DF 00 FF FF FF 00 00 00 00 00 02 2F 00 00
00064DE8 04 50 00 00 06 70 00 00 08 90 00 00 0A B0 00 00

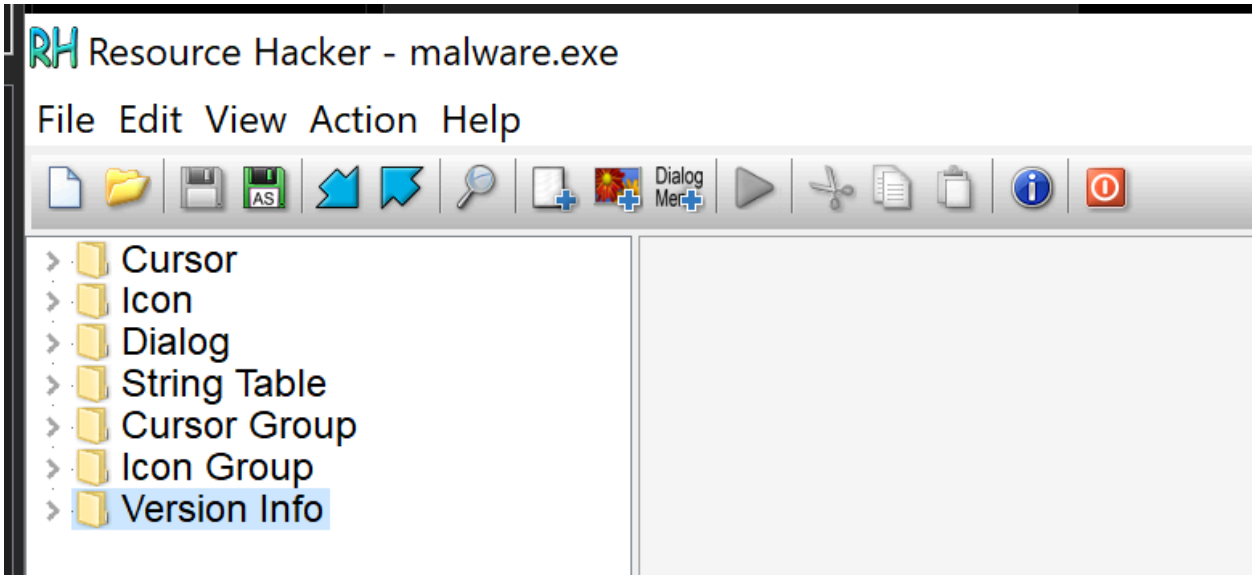

ICON 000B 0418
ICON 000C 0418
ICON 000D 0418
GROUP_ICON 0097 0418
ICON 000E 0418
ICON 000F 0418
ICON 0010 0418
ICON 0011 0418
ICON 0012 0418
ICON 0013 0418
ICON 0014 0418
ICON 0015 0418
GROUP_ICON 009B 0418
ICON 0016 0418
ICON 0017 0418
ICON 0018 0418
ICON 0019 0418
ICON 001A 0418
ICON 001B 0418
ICON 001C 0418
ICON 001D 0418
GROUP_ICON 009F 0418
CURSOR 001E 0000
CURSOR 001F 0000
CURSOR 0020 0000
GROUP_CURSOR 094E 0000
CURSOR 0021 0000
CURSOR 0022 0000
GROUP_CURSOR 0953 0000
CURSOR 0023 0000
CURSOR 0024 0000
CURSOR 0025 0000
GROUP_CURSOR 0954 0000
VERSION 0001 0000
pFile Raw Data Value
000679A8 EC 01 34 00 00 00 56 00 53 00 5F 00 56 00 45 00
000679B8 52 00 53 00 49 00 4F 00 4E 00 5F 00 49 00 4E 00
000679C8 46 00 4F 00 00 00 00 00 BD 04 EF FE 00 00 01 00
000679D8 00 00 2A 00 00 00 00 00 00 00 34 00 00 00 00 00
000679E8 0A 95 00 00 00 00 00 00 23 08 02 00 00 00 00 00
000679F8 00 00 00 00 00 00 00 00 00 00 00 00 4A 01 00 00
00067A08 01 00 53 00 74 00 72 00 69 00 6E 00 67 00 46 00
00067A18 69 00 6C 00 65 00 49 00 6E 00 66 00 6F 00 00 00
00067A28 26 01 00 00 01 00 30 00 34 00 32 00 39 00 32 00
00067A38 34 00 45 00 36 00 00 00 38 00 0C 00 01 00 46 00
00067A48 69 00 6C 00 65 00 56 00 65 00 72 00 73 00 69 00
00067A58 6F 00 6E 00 00 00 00 00 32 00 34 00 2E 00 33 00
00067A68 34 00 2E 00 32 00 38 00 2E 00 37 00 39 00 00 00
00067A78 32 00 05 00 01 00 46 00 69 00 6C 00 65 00 44 00
00067A88 65 00 73 00 63 00 72 00 69 00 70 00 74 00 69 00
00067A98 6F 00 6E 00 00 00 00 00 42 00 69 00 6C 00 6C 00
00067AA8 00 00 00 00 34 00 06 00 01 00 4F 00 72 00 69 00
00067AB8 67 00 69 00 6E 00 61 00 6C 00 46 00 69 00 6C 00
00067AC8 65 00 6E 00 61 00 6D 00 65 00 00 00 46 00 69 00
00067AD8 72 00 65 00 73 00 00 00 30 00 08 00 01 00 50 00
00067AE8 72 00 6F 00 64 00 75 00 63 00 74 00 4E 00 61 00
00067AF8 6D 00 65 00 00 00 00 00 53 00 6D 00 65 00 6C 00
00067B08 66 00 69 00 65 00 00 00 3E 00 0B 00 01 00 50 00
00067B18 72 00 6F 00 64 00 75 00 63 00 74 00 56 00 65 00
00067B28 72 00 73 00 69 00 6F 00 6E 00 73 00 00 00 00 00
00067B38 35 00 2E 00 33 00 30 00 2E 00 34 00 30 00 2E 00
00067B48 39 00 35 00 00 00 00 00 44 00 00 00 01 00 56 00
00067B58 61 00 72 00 46 00 69 00 6C 00 65 00 49 00 6E 00
00067B68 66 00 6F 00 00 00 00 00 24 00 04 00 00 00 54 00
00067B78 72 00 61 00 6E 00 73 00 6C 00 61 00 74 00 69 00
00067B88 6F 00 6E 00 00 00 00 00 0A 16 E3 04



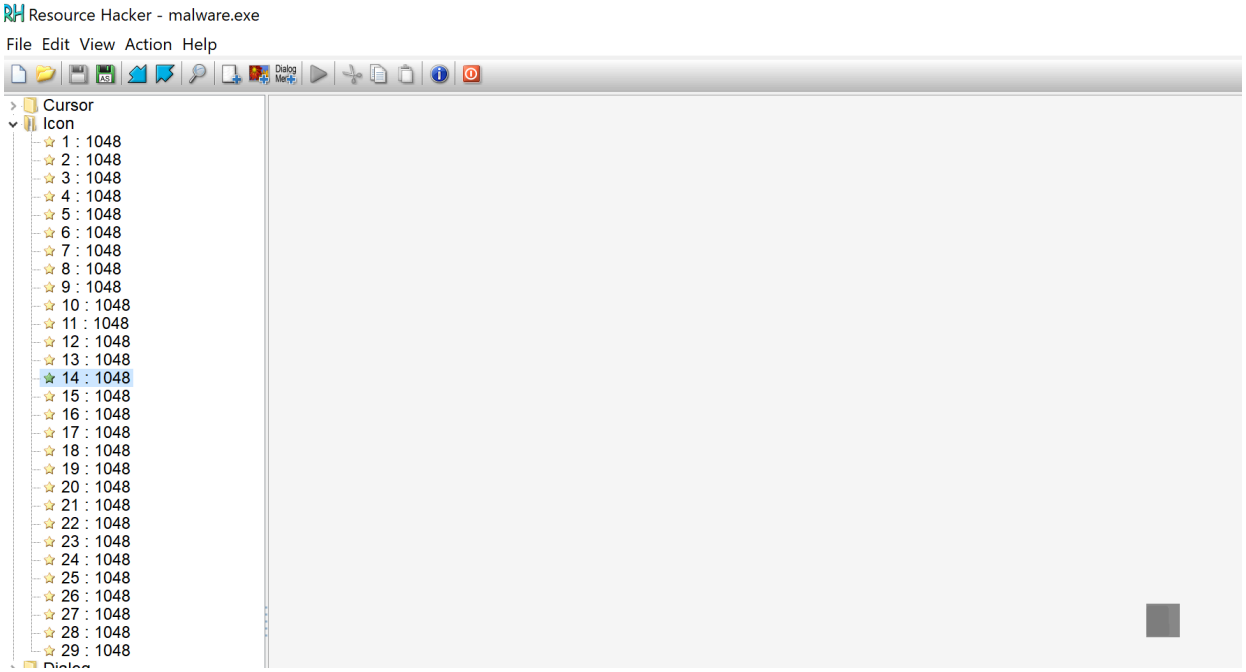
The ICON, GROUP, CURSOR, VERSION, and STRING include information for the pFile, Raw Data, and Value. All of these sections are basically just hex dumps with ASCII values under the Value section. Since they are in hex, I am unable to read information and data.

Overall, PEview showed me a lot of useful information regarding the malware I am attempting to reverse. While looking through PEview, I found a table of all of the functions used by this malware, as well as all of the dll files used. This is very helpful, as it allows me to understand more about Lumma Stealer and what exactly it does. I found icons and strings in the form of hex dumps. Due to these icons and strings being in hex, I am unable to read them and see exactly what they are or do. A majority of the information in PEview, however, was in the form of hex dumps which made it unreadable. Since this information is in hex, I can use Resource Hacker to help me make more sense of this information. I will now use Resource Hacker to view the information and dive even deeper into this malware and what it does.

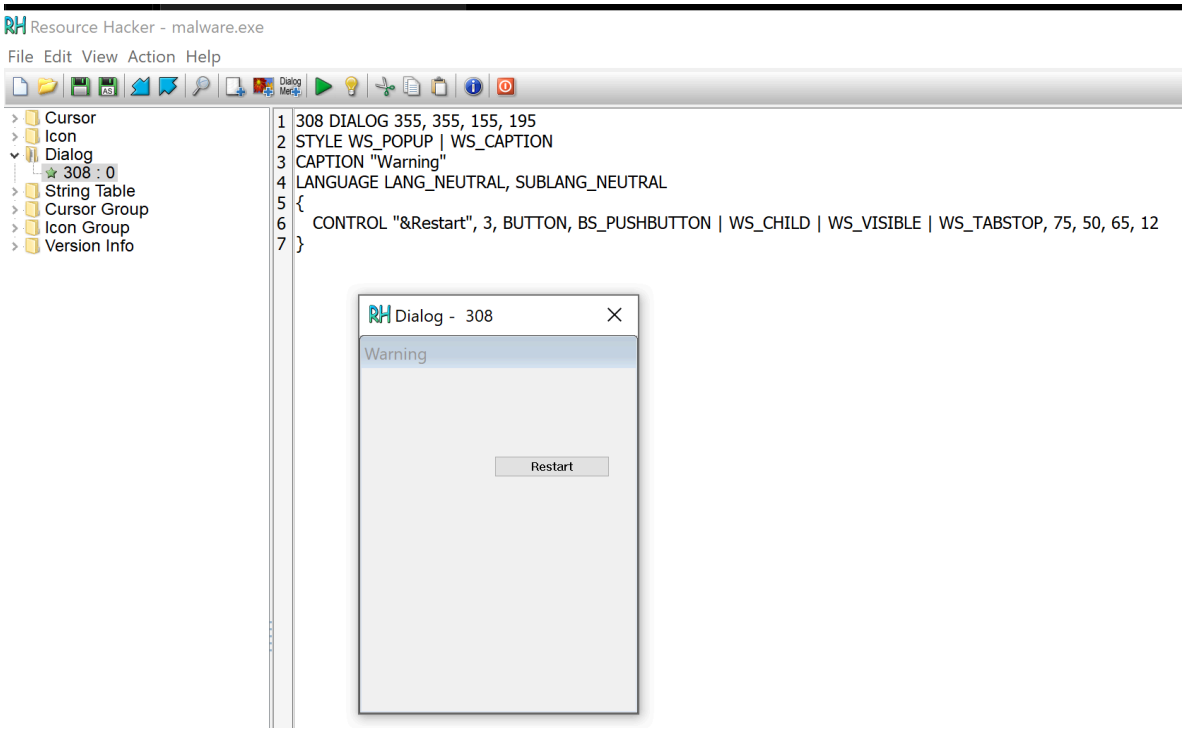
Resource Hacker:



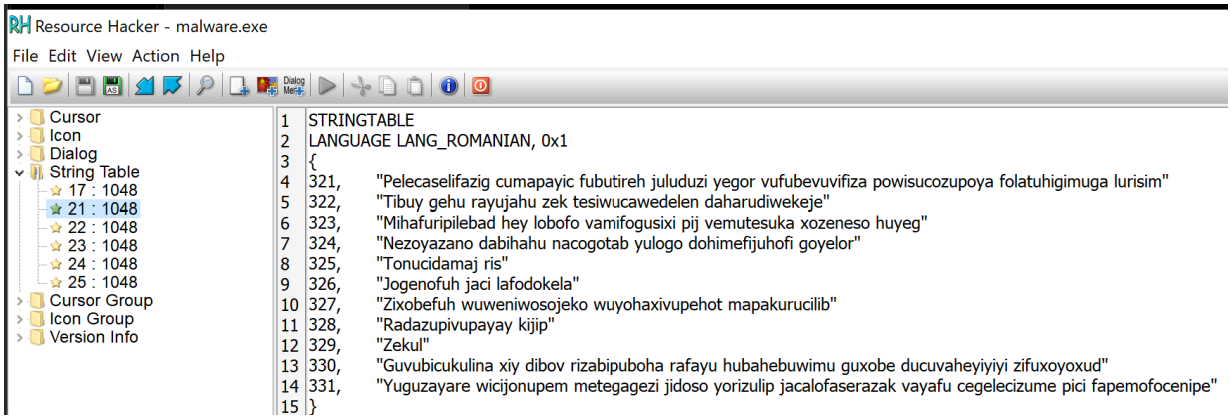
The screenshot above shows that I have opened the malware executable in Resource Hacker. Once opened, I notice that the folders it shows in Resource Hacker are the same subsections that were shown in PEview under the SECTION.rsrc section. Below I included a screenshot from each section.



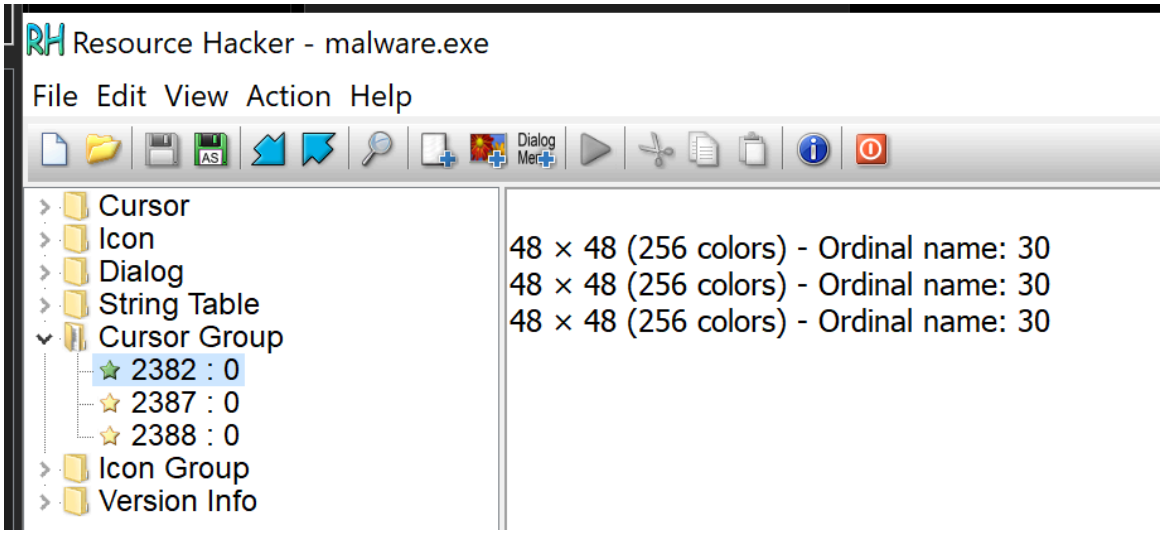
The screenshot above is a screenshot of one of the files in the Icon folder. Each file in this folder has different images of cubes that are all different shades of gray.



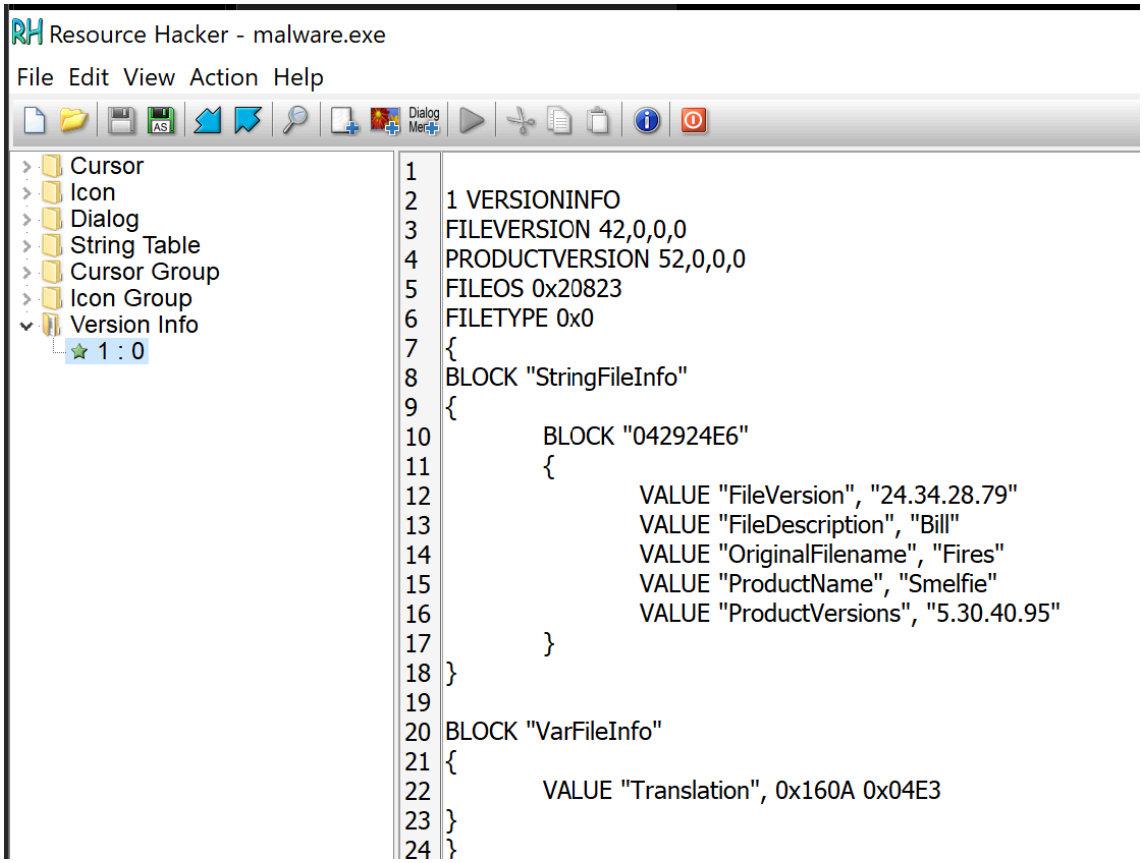
This screenshot shows the file in the Dialog folder. In this file is what seems to be code. This code creates a pop-up with a warning message and a restart button.



This screenshot is of a file in the String Table folder. All of these files in this folder seem to be in Romanian. Each line in the files is a new string, or something new written in Romanian.



The files in the Cursor Group folder seem to have a “number x number” which correlates to a certain number of colors and an ordinal name.

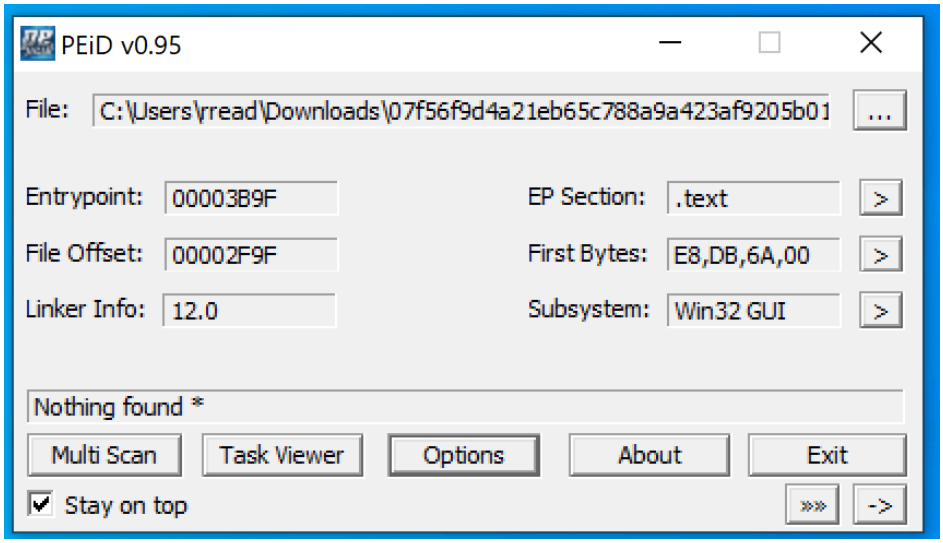


The file in the Version info as shown above gives information regarding the version of a file. It gives a value for the version, description, filename, product name, and product version.

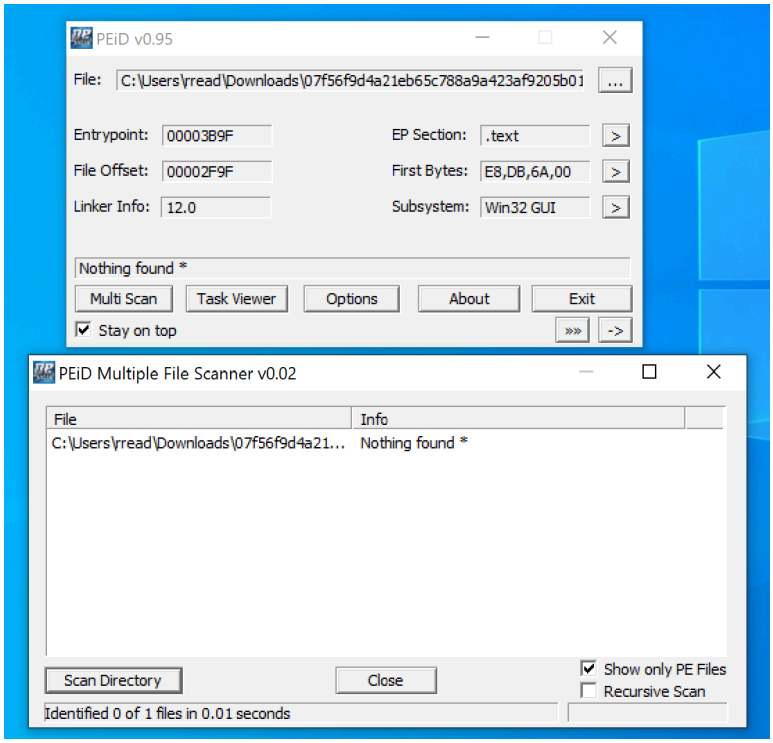
Overall when looking at what Resource Hacker provides, it seems to go more in depth about certain files that are unreadable from PEview. Some files in PEview were just hex dumps with

random ASCII values printed to the side. In Resource Hacker, you can now see those hex dumps in readable code and images. For example, you are now able to see icons instead of hex, dialog or pop-up boxes, strings in what seem to be foreign languages, and other code. This allows for a better understanding of what these files entail, and what this malware is capable of.

PEiD:



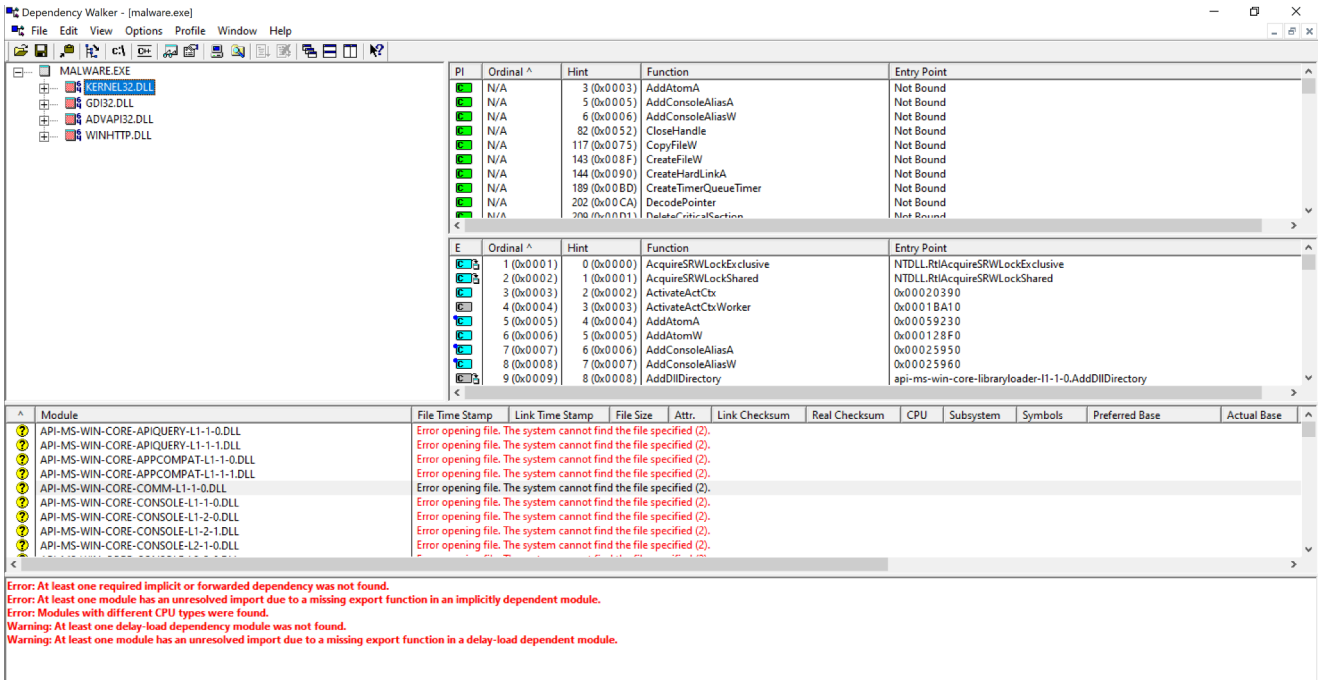
The above screenshot shows my malware uploaded to PEiD. On the main screen PEiD gives some information regarding Lumma Stealer such as file offset, first bytes, subsystem and more. PEiD also allows you to scan the file that you upload to see if anything is found regarding the compiler used.



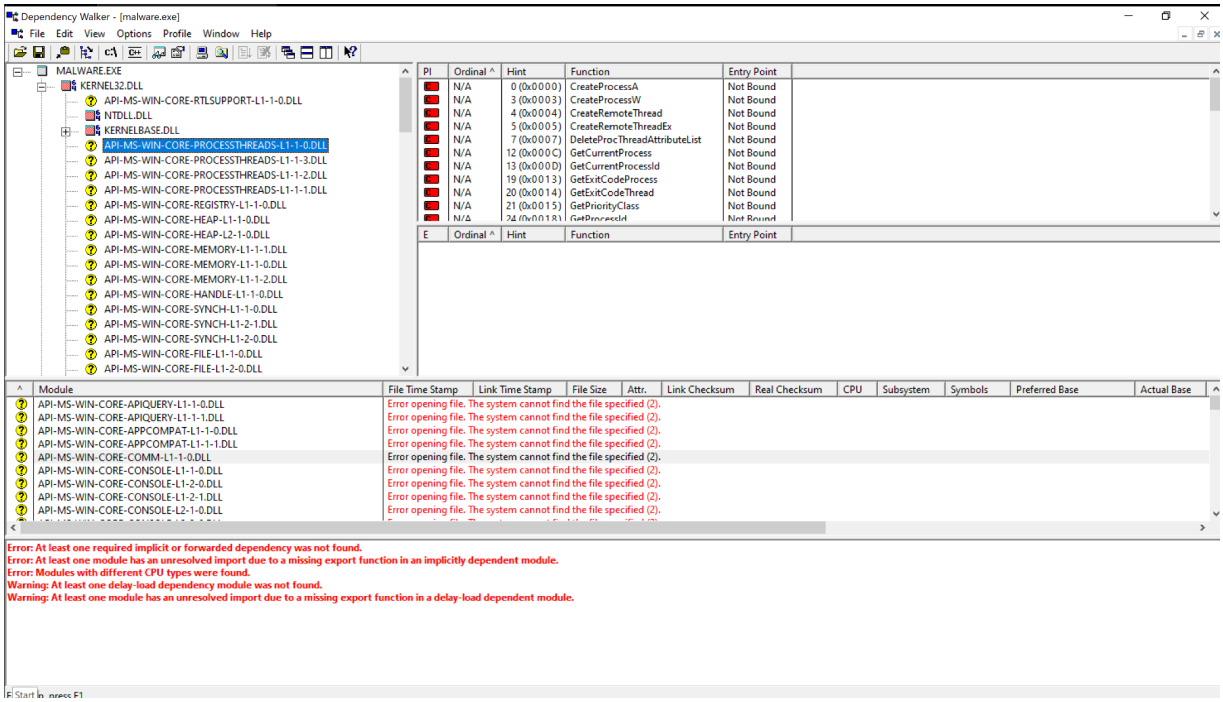
The above screenshot shows me running a scan on the malware executable that I uploaded to PEiD. I ran the scan multiple times and every time it stated nothing was found. This could be because the threat actor that wrote this malware took precautions in hiding or burying what they didn't want to be found, or it could just be that PEiD is unsure of the compiler that was used.

The information provided from PEiD was somewhat helpful. I did get some new information I hadn't found before. One thing that was super helpful in PEiD was the additional information it provided for EP Section, First Bytes, and Subsystem. The additional information for First Bytes gave me code in assembly and also allowed me to see the strings that were used in this code and where they were used. This furthered my understanding regarding the malware.

Dependency Walker:



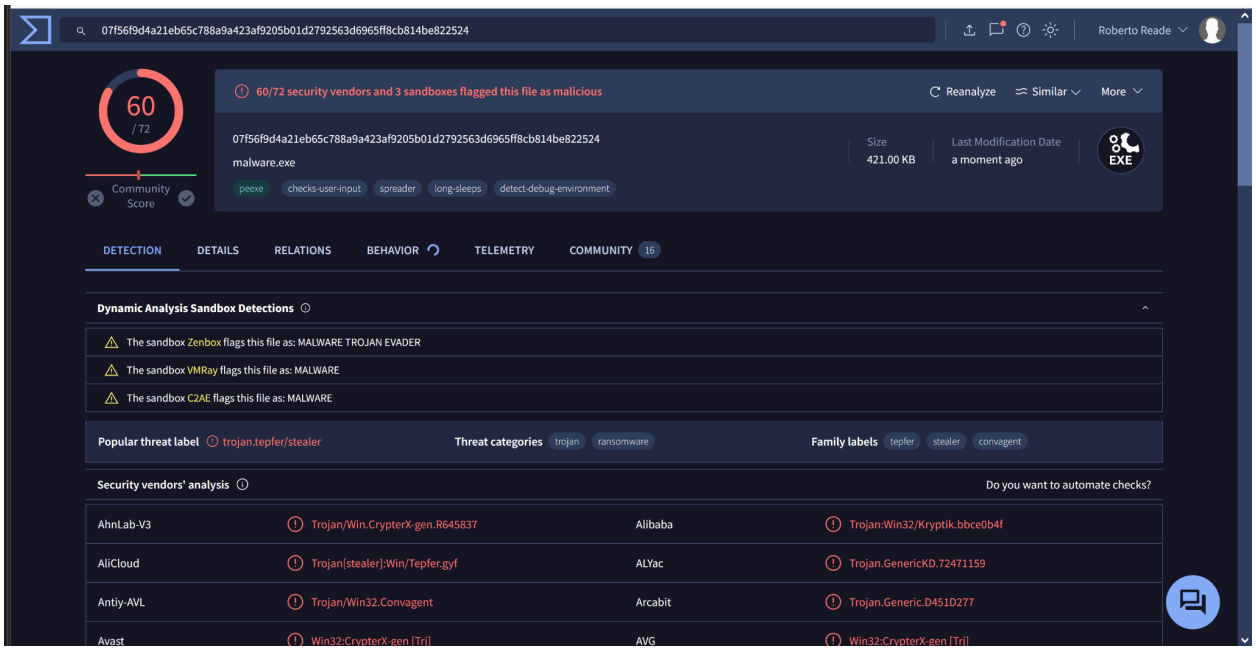
The screenshot above shows my malware executable opened in Dependency Walker. Dependency Walker shows the .DLL files that are used in Lumma Stealer as well as the API's and functions used in those .DLL files. There is also information regarding errors where some files may be missing.



This screenshot is very similar to the last one, however, I opened an API that was in one of the .DLL files instead of just looking at the .DLL file itself.

Overall, Dependency Walker was very helpful and provided good information regarding the malware I am using. It provided me with the .DLL files that were used. These include KERNEL32.DLL, GDI32.DLL, ADVAPI32.DLL, and WINHTTP.DLL. It also goes in depth about the .DLL files used within these .DLL files, and the API's used within them as well. For each .DLL and API, it gave all the functions used within them. It also gave entry points, offsets and what seemed to be line numbers for each function. Reviewing these .DLL, API, and functions gave me a very good idea of what Lumma Stealer does and what it can be used for. There were a lot of functions including HTTP as well as getter and setter functions. Dependency Walker also listed errors when it noticed files were missing from certain API's.

VirusTotal:

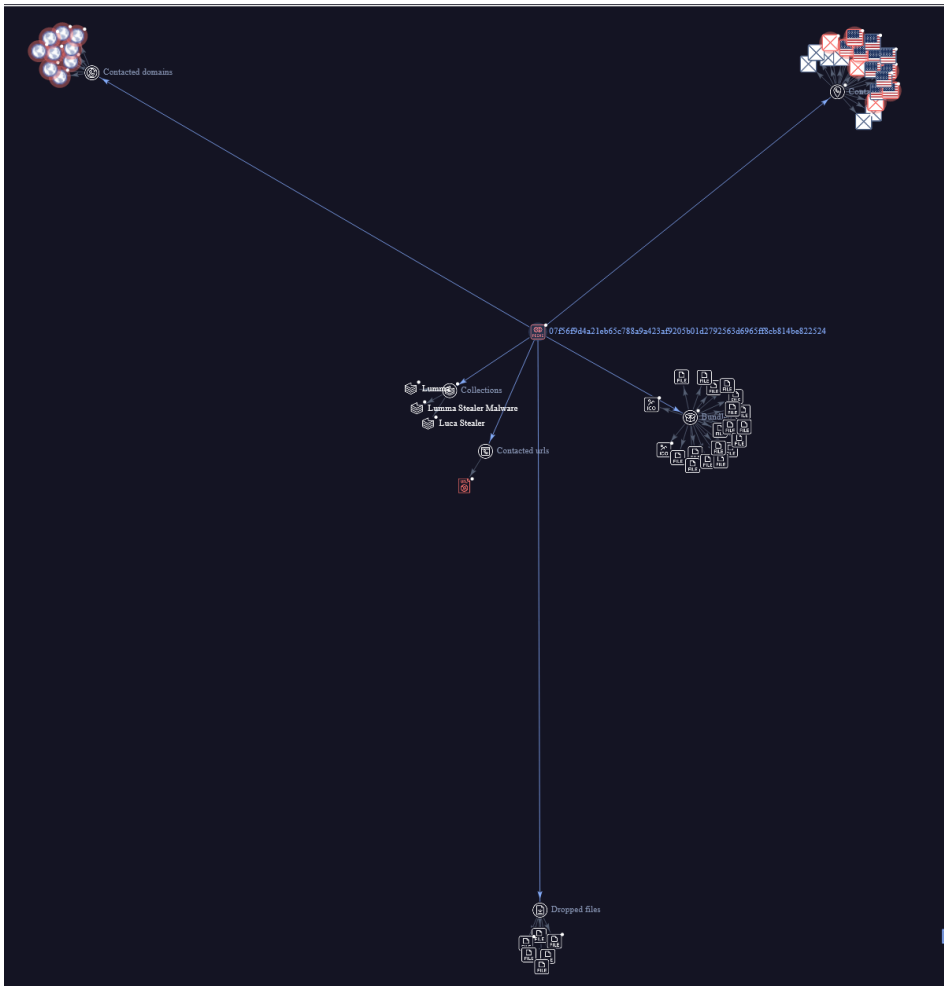


The screenshot above shows that I opened the malware I am using in VirusTotal. Once I opened it and VirusTotal checked the hash, I was able to see that a majority of security vendors and sandboxes flagged the file as malicious. To be more specific, 60 out of 72 vendors, and 3 sandboxes found the file to be malicious.

VirusTotal has multiple tabs you can click on to learn more about the file that was uploaded. When I went through the detection tab, I was able to see what vendors thought the file was malicious and what vendors thought it was safe.

In the details tab, I am able to see a bunch of information. Some of which I already know from previous tools and some new information. I am able to see things such as properties of the file which include the hashes and file type. I am also able to see the previous names, and history of the file. Lastly in this section I can see executable information such as compilers used, header, sections of data, and imports.

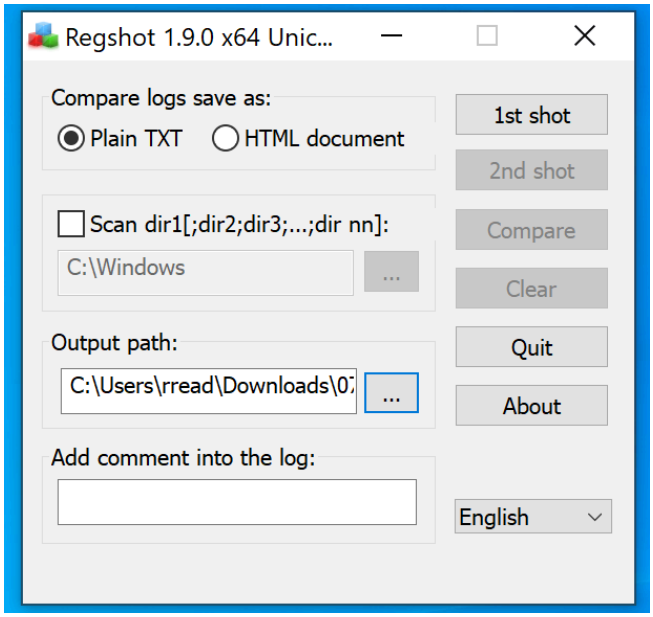
While looking into the relations tab I noticed some contacted URLs, domains and IP addresses that Lumma Stealer has probably been used on. I am also able to see a section of bundled and deleted files, however there is not a lot of information in these sections. Lastly in this section I can open a graph summary to show all the relations of this malware. The graph summary screenshot is shown below.



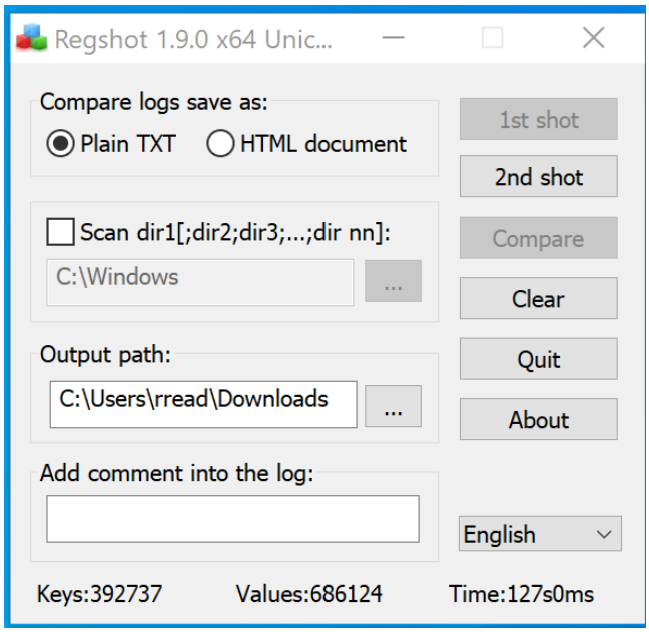
When I moved onto the behavior tab, I found some interesting information regarding the behavior of the malware. The three main things that were listed first in the behavior tab was that this malware checks user input, detects debug environments, and sleeps for long periods of time. It goes more in depth in these behaviors as well as their capabilities. After this was listed, it went on to the network communication of this malware. After the network communication, there was information regarding the file system, registry, and process actions. It was very interesting to see what Lumma Stealer can do over a network and on someone's device.

Basic Dynamic Analysis:

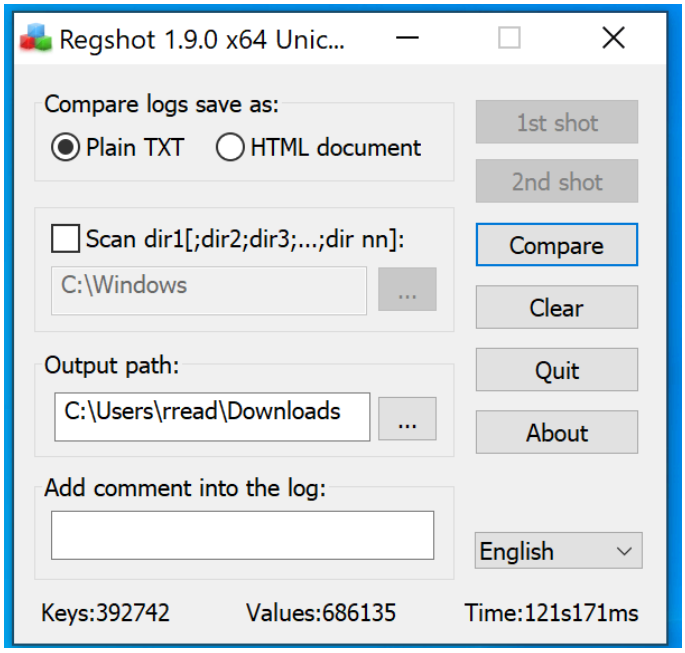
Regshot:



Above is a screenshot of me making my first shot in Regshot. I set the compare logs to plain text and I set the output path to my Downloads folder since that is where the malware file is located.



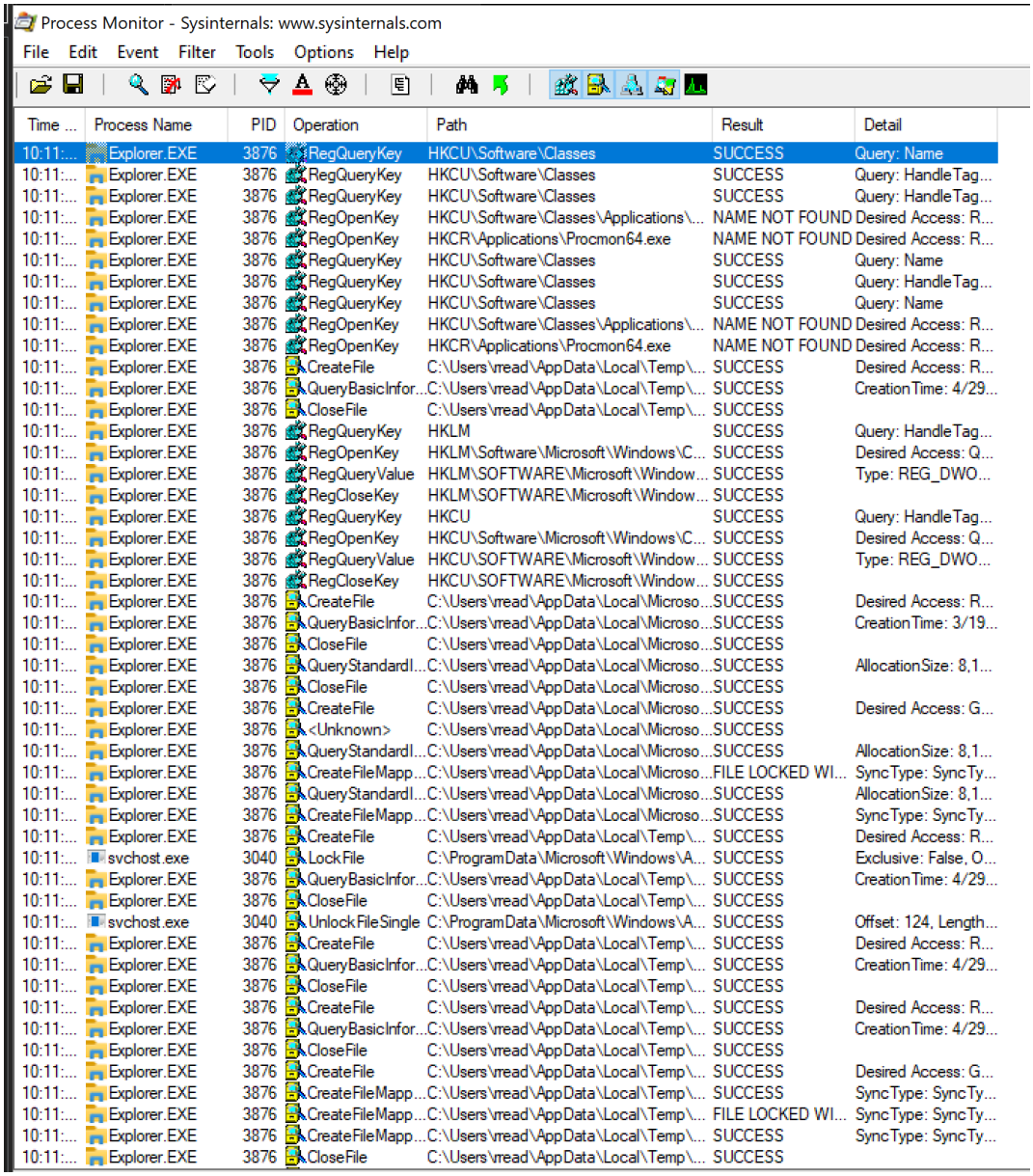
The above screenshot is me taking the second shot in Regshot. Between shots 1 and 2, I ran the malware executable file. This way when I compare them, I am able to see what exactly Lumma Stealer did.



The two screenshots above show the comparison between the first and second shot in Regshot. While looking at what was modified, I realized that the HKLM registry was deleted, and some other interesting HKLM registry keys were added. I also realized that there were long values that seemed to be in hex that were added or modified on my system. As I looked into what these hex values went to, I realized that they were values that were added or modified for weird looking functions and weird paths to different directories. As I looked deeper into these directories, I realized that Microsoft Explorer was opened, and information is being gathered about this browser.

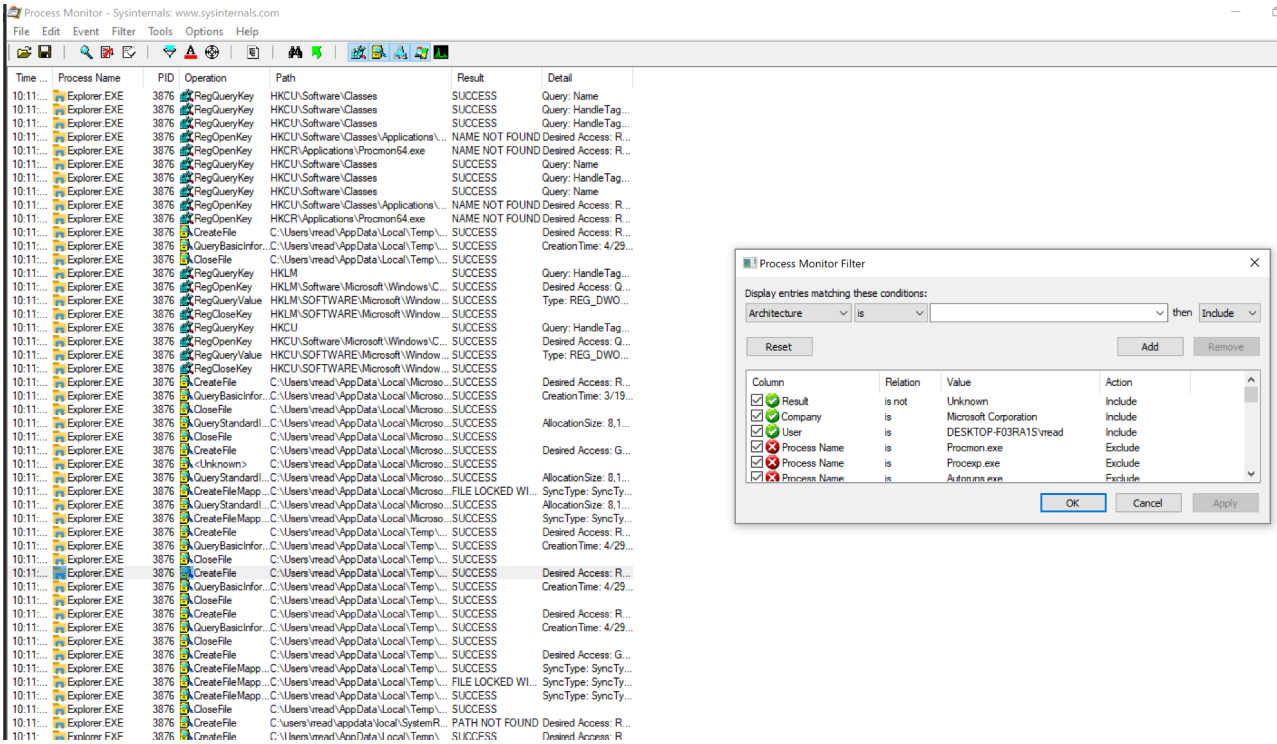
Overall, Regshot was very helpful because it allowed me to see what Lumma Stealer is doing. I can now see that this malware deletes and adds registry keys on the device. It also runs functions in the background that gather information about the device as well as data on browsers and browser extensions.

Process Monitor:



Time ...	Process Name	PID	Operation	Path	Result	Detail
10:11:...	Explorer.EXE	3876	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
10:11:...	Explorer.EXE	3876	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTag...
10:11:...	Explorer.EXE	3876	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTag...
10:11:...	Explorer.EXE	3876	RegOpenKey	HKCU\Software\Classes\Applications\...	NAME NOT FOUND	Desired Access: R...
10:11:...	Explorer.EXE	3876	RegOpenKey	HKCR\Applications\Procmon64.exe	NAME NOT FOUND	Desired Access: R...
10:11:...	Explorer.EXE	3876	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
10:11:...	Explorer.EXE	3876	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: HandleTag...
10:11:...	Explorer.EXE	3876	RegQueryKey	HKCU\Software\Classes	SUCCESS	Query: Name
10:11:...	Explorer.EXE	3876	RegOpenKey	HKCU\Software\Classes\Applications\...	NAME NOT FOUND	Desired Access: R...
10:11:...	Explorer.EXE	3876	RegOpenKey	HKCR\Applications\Procmon64.exe	NAME NOT FOUND	Desired Access: R...
10:11:...	Explorer.EXE	3876	CreateFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	Desired Access: R...
10:11:...	Explorer.EXE	3876	QueryBasicInfor...	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	CreationTime: 4/29...
10:11:...	Explorer.EXE	3876	CloseFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	
10:11:...	Explorer.EXE	3876	RegQueryKey	HKLM	SUCCESS	Query: HandleTag...
10:11:...	Explorer.EXE	3876	RegOpenKey	HKLM\Software\Microsoft\Windows\C...	SUCCESS	Desired Access: Q...
10:11:...	Explorer.EXE	3876	RegQueryValue	HKLM\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
10:11:...	Explorer.EXE	3876	RegCloseKey	HKLM\SOFTWARE\Microsoft\Window...	SUCCESS	
10:11:...	Explorer.EXE	3876	RegQueryKey	HKCU	SUCCESS	Query: HandleTag...
10:11:...	Explorer.EXE	3876	RegOpenKey	HKCU\Software\Microsoft\Windows\C...	SUCCESS	Desired Access: Q...
10:11:...	Explorer.EXE	3876	RegQueryValue	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Type: REG_DWO...
10:11:...	Explorer.EXE	3876	RegCloseKey	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	
10:11:...	Explorer.EXE	3876	CreateFile	C:\Users\rread\AppData\Local\Microso...	SUCCESS	Desired Access: R...
10:11:...	Explorer.EXE	3876	QueryBasicInfor...	C:\Users\rread\AppData\Local\Microso...	SUCCESS	CreationTime: 3/19...
10:11:...	Explorer.EXE	3876	CloseFile	C:\Users\rread\AppData\Local\Microso...	SUCCESS	
10:11:...	Explorer.EXE	3876	QueryStandardI...	C:\Users\rread\AppData\Local\Microso...	SUCCESS	AllocationSize: 8,1...
10:11:...	Explorer.EXE	3876	CloseFile	C:\Users\rread\AppData\Local\Microso...	SUCCESS	
10:11:...	Explorer.EXE	3876	CreateFile	C:\Users\rread\AppData\Local\Microso...	SUCCESS	Desired Access: G...
10:11:...	Explorer.EXE	3876	<Unknown>	C:\Users\rread\AppData\Local\Microso...	SUCCESS	
10:11:...	Explorer.EXE	3876	QueryStandardI...	C:\Users\rread\AppData\Local\Microso...	SUCCESS	AllocationSize: 8,1...
10:11:...	Explorer.EXE	3876	CreateFileMapp...	C:\Users\rread\AppData\Local\Microso...	FILE LOCKED WI...	SyncType: SyncTy...
10:11:...	Explorer.EXE	3876	QueryStandardI...	C:\Users\rread\AppData\Local\Microso...	SUCCESS	AllocationSize: 8,1...
10:11:...	Explorer.EXE	3876	CreateFileMapp...	C:\Users\rread\AppData\Local\Microso...	SUCCESS	SyncType: SyncTy...
10:11:...	Explorer.EXE	3876	CreateFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	Desired Access: R...
10:11:...	svchost.exe	3040	LockFile	C:\ProgramData\Microsoft\Windows\A...	SUCCESS	Exclusive: False, O...
10:11:...	Explorer.EXE	3876	QueryBasicInfor...	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	CreationTime: 4/29...
10:11:...	Explorer.EXE	3876	CloseFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	
10:11:...	svchost.exe	3040	UnlockFileSingle	C:\ProgramData\Microsoft\Windows\A...	SUCCESS	Offset: 124, Length...
10:11:...	Explorer.EXE	3876	CreateFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	Desired Access: R...
10:11:...	Explorer.EXE	3876	QueryBasicInfor...	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	CreationTime: 4/29...
10:11:...	Explorer.EXE	3876	CloseFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	
10:11:...	Explorer.EXE	3876	CreateFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	Desired Access: R...
10:11:...	Explorer.EXE	3876	QueryBasicInfor...	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	CreationTime: 4/29...
10:11:...	Explorer.EXE	3876	CloseFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	
10:11:...	Explorer.EXE	3876	CreateFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	Desired Access: G...
10:11:...	Explorer.EXE	3876	CreateFileMapp...	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	SyncType: SyncTy...
10:11:...	Explorer.EXE	3876	CreateFileMapp...	C:\Users\rread\AppData\Local\Temp\...	FILE LOCKED WI...	SyncType: SyncTy...
10:11:...	Explorer.EXE	3876	CreateFileMapp...	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	SyncType: SyncTy...
10:11:...	Explorer.EXE	3876	CloseFile	C:\Users\rread\AppData\Local\Temp\...	SUCCESS	

When I opened Process Monitor, I decided to run the malware that I am using. When I did this, I noticed a ton of Explorer.exe processes pop-up, as well as some Windows executables that are used for virus protection, and error logging. I decided to add filters to narrow down the long list of processes. I included a screenshot below showing the filters I implemented.



I implemented three filters. When implementing these filters I decided to make them a little more broad so I wouldn't filter out any important processes.

The first filter I implemented was if the user is rread then include. I did this because this user is me, and I am the one that ran the malware, so I only wanted to see processes used under this user.

The next filter I used was if the company is Microsoft Corporation then include. I did this because every time I ran this malware, in this tool and others, Explorer.EXE always opens and there are operations done on it. Since this specific malware targets browsers, I decided to include the company that owns this browser.

The last filter I used was if the result is not unknown then include. I did this because when I was looking through the processes before I implemented the filters, I saw most of the results were success with some being name not found. I didn't notice any processes that went along with this malware that had the result of unknown. Therefore, I decided to include everything that was not a result of unknown.

After implementing all of these filters, I went back and looked through the processes. At first I noticed that Explorer.EXE was listed a majority of the time. I noticed for these processes that there were operations that queried registry keys, and a lot of create, call, and query operations used in Explorer.EXE.

As I kept scrolling down, I noticed WerFault.exe was a running process. I researched what exactly this executable is. Apparently it is used for Windows error reporting, however, hackers use this executable to deploy malware.

Looking further, I noticed that backgroundtaskhost.exe was running multiple processes. These processes all had operations on registry keys. When I looked at the properties of some of these processes, they all had weird looking command lines. It was all just strings of random letters. This seemed very odd.

After that, I noticed processes from RuntimeBroker.exe. I did some research on this executable as well and found that it manages permissions for PC apps from the Microsoft store. I looked at the properties of these processes and saw that they all had to do with embedding.

Overall, looking into Process Monitor and using filters help me gain more information regarding Lumma Stealer. It allowed me to understand what is being done when Explorer.EXE is opened, and what processes are running in the background.

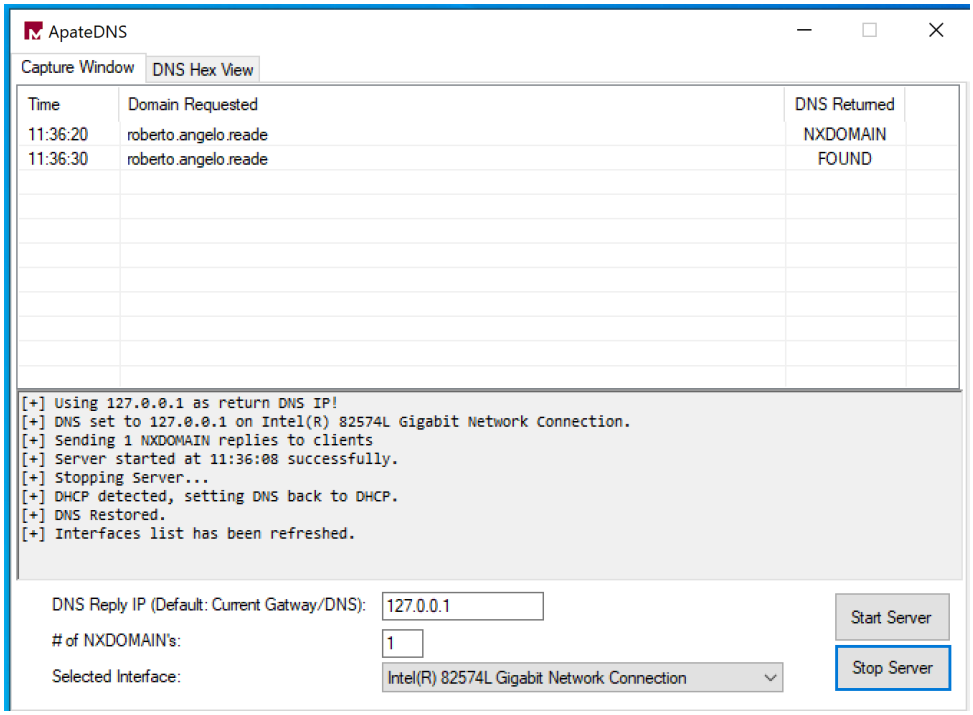
Process Explorer:

After opening Process Explorer, I am able to see a majority of the processes I noticed in Process Monitor. Process Explorer helped me understand how much of the CPU these

processes are using as well as the amount of bytes. Process Explorer gives you the PID of each process and a description, however, it does not go into as much detail as Process Monitor. You are not able to see the path, the operations, or the stack for each process. Overall Process Explorer helped me understand how much memory each process was taking up. On the other hand, it did not further my knowledge of what this malware is capable of doing.

ApateDNS:

Since the Lumma Stealer does not do anything with the network or internet, I am unable to use ApateDNS on the malware. I did however start an ApateDNS server and do a couple of my own tests on it as shown below.



When I opened ApateDNS, I put in the loopback address for the DNS Reply IP, and I set the number of NXDOMAIN's to 1. I then started the server. As you can see above I tried to ping my name. The first time it returned an NXDOMAIN and the second time it found the DNS. This is because I specified the number of NXDOMAIN's to be 1.

```
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\rread>ping roberto.angelo.reade
Ping request could not find host roberto.angelo.reade. Please check the name and try again.

C:\Users\rread>ping roberto.angelo.reade

Pinging roberto.angelo.reade [127.0.0.1] with 32 bytes of data:
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128
Reply from 127.0.0.1: bytes=32 time<1ms TTL=128

Ping statistics for 127.0.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\rread>
```

This screenshot above shows what happened when I tried to ping my name. The first time it could not find this host, and the second time I attempted to ping my name I got a response. Again, as stated above, this was due to the fact that I set the number of NXDOMAIN's to be 1.

```
C:\Users\rread>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : DESKTOP-F03RA1S
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : localdomain

Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . : localdomain
   Description . . . . . . . . . . . : Intel(R) 82574L Gigabit Network Connection
   Physical Address. . . . . . . . . : 00-0C-29-62-55-4C
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::3fa3:5916:9746:300%3(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.115.128(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Monday, April 29, 2024 9:50:42 AM
   Lease Expires . . . . . . . . . . : Monday, April 29, 2024 12:06:08 PM
   Default Gateway . . . . . . . . . : 192.168.115.2
   DHCP Server . . . . . . . . . . . : 192.168.115.254
   DHCPv6 IAID . . . . . . . . . . . : 100666409
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2D-8B-A3-2A-00-0C-29-62-55-4C
   DNS Servers . . . . . . . . . . . : 127.0.0.1
   Primary WINS Server . . . . . . . : 192.168.115.2
```

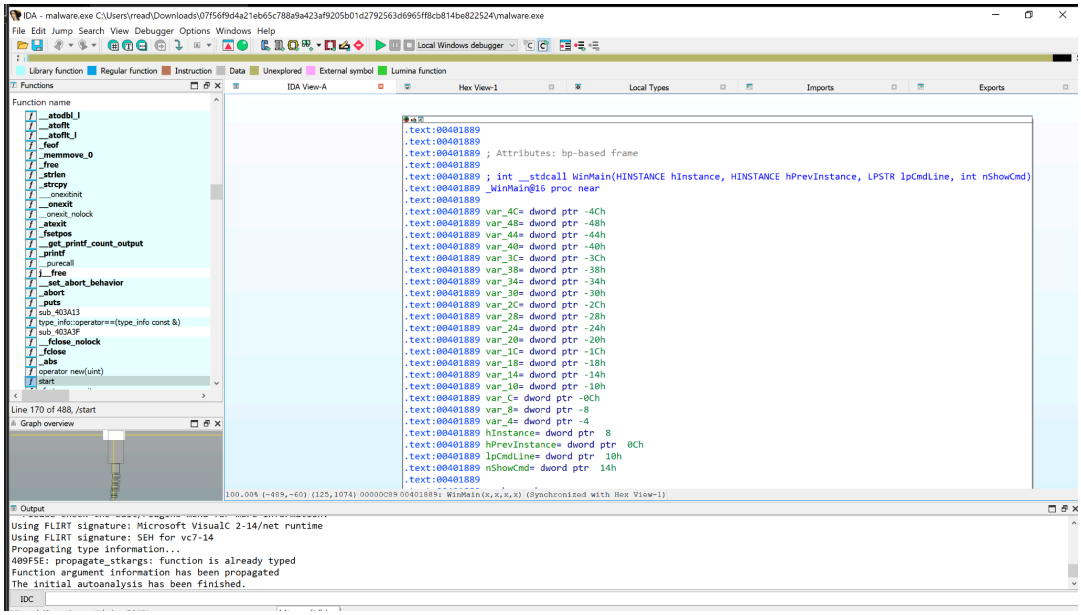
The screenshot above shows me entering the ipconfig /all command while the ApateDNS server is running. As you can see, the DNS server is set to the loopback address of 127.0.0.1. This is because in the server settings I set the DNS Reply IP to be the loopback address.

```
C:\Users\rread>netstat -an | more

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
  TCP    0.0.0.0:5040           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:7680           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49664          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49665          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49666          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49667          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49668          0.0.0.0:0              LISTENING
  TCP    0.0.0.0:49679          0.0.0.0:0              LISTENING
  TCP    192.168.115.128:139    0.0.0.0:0              LISTENING
  TCP    192.168.115.128:49687  52.159.126.152:443     ESTABLISHED
  TCP    192.168.115.128:50096  184.87.60.157:80       TIME_WAIT
  TCP    192.168.115.128:50099  204.79.197.200:443     ESTABLISHED
  TCP    192.168.115.128:50100  20.42.73.25:443        ESTABLISHED
  TCP    192.168.115.128:50101  13.107.237.254:443     ESTABLISHED
  TCP    192.168.115.128:50102  4.150.240.254:443      ESTABLISHED
  TCP    192.168.115.128:50103  13.107.226.40:443      CLOSE_WAIT
  TCP    192.168.115.128:50104  204.79.197.222:443     ESTABLISHED
  TCP    192.168.115.128:50105  184.28.190.72:443      ESTABLISHED
  TCP    [::]:135               [::]:0                 LISTENING
  TCP    [::]:445               [::]:0                 LISTENING
  TCP    [::]:7680              [::]:0                 LISTENING
  TCP    [::]:49664             [::]:0                 LISTENING
  TCP    [::]:49665             [::]:0                 LISTENING
  TCP    [::]:49666             [::]:0                 LISTENING
  TCP    [::]:49667             [::]:0                 LISTENING
  TCP    [::]:49668             [::]:0                 LISTENING
  TCP    [::]:49679             [::]:0                 LISTENING
  UDP    0.0.0.0:53             *:*
```

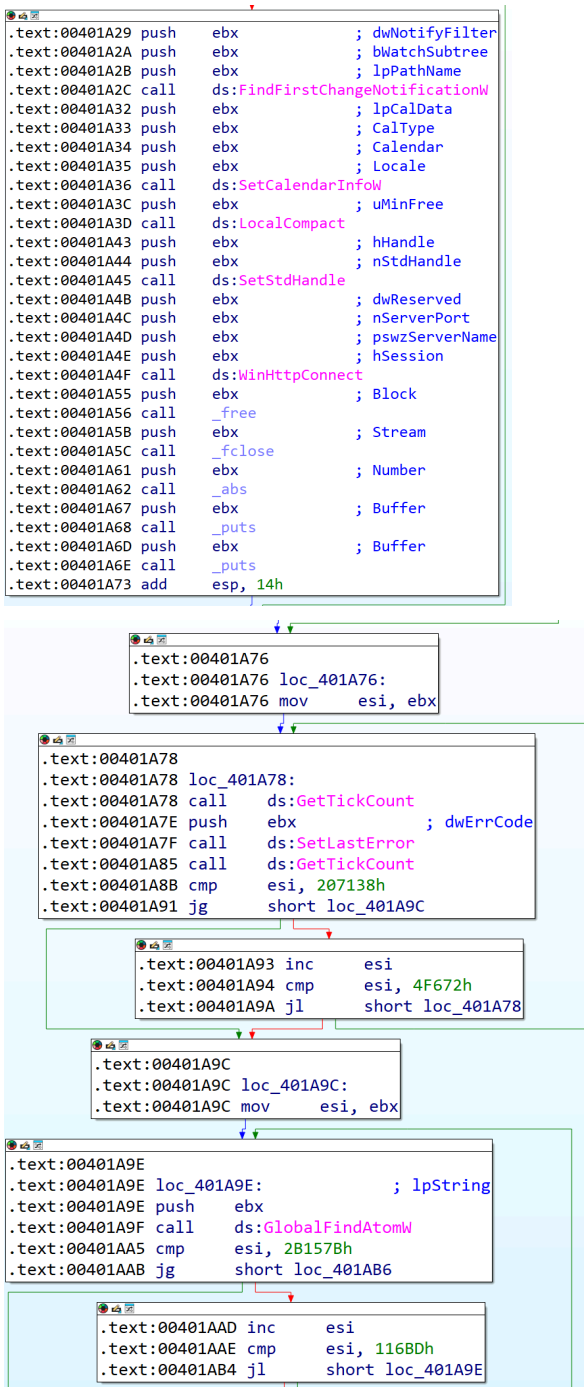
The final step I did with ApateDNS was check to see if port 53 was present in the “netstat -an | more” command. In the screenshot above, you can see that it is in fact present. This means that this device can send traffic to the DNS server listening on port 53. If the ApateDNS server was not running, then port 53 would not be present in the output of the “netstat -an | more” command.

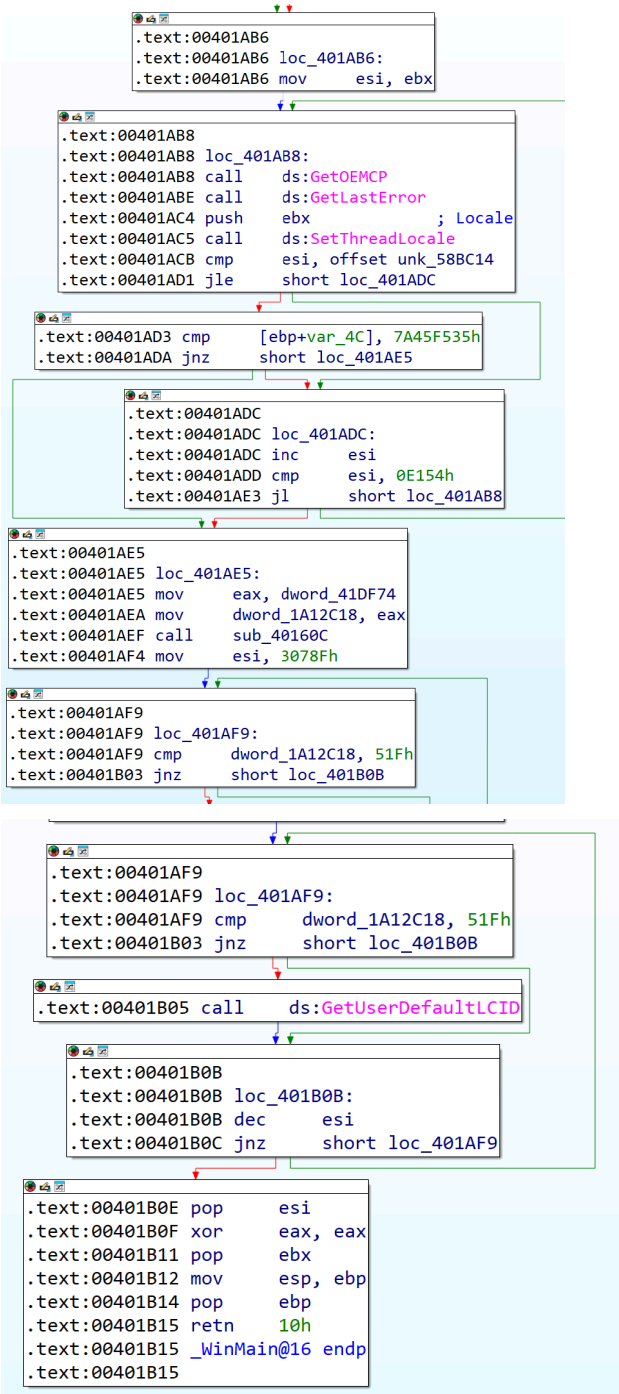
Advanced Static Analysis:

IDA:



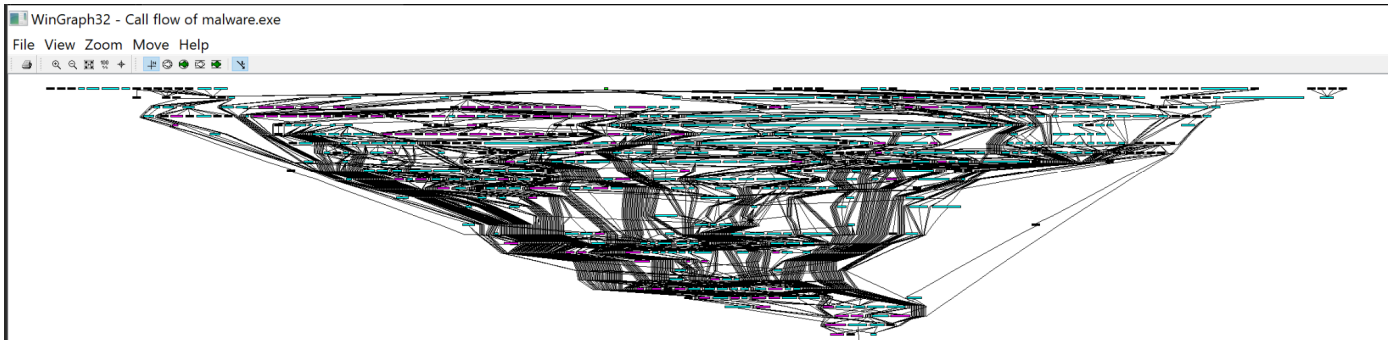
The screenshot above shows me opening the malware in IDA. When I did this, I was brought right to the main function. In this function, there are many pointer variables set to hex values. The rest of the main function seems to just be a lot of add, sub, mul, and mov commands. Overall, the main function seems to be pretty basic. Below the main function are other functions that include if and for loops. All of these functions have a call to other functions. Screenshots of these functions are shown below.





The four screenshots above are the functions listed under main. In these, there are calls to other functions that will execute the malware. When I click on these function calls, I am able to see the variables that are passed into each function as well as what that specific function returns.

Since I saw so many function calls, I decided to look at the graph of the call flowchart of this malware.



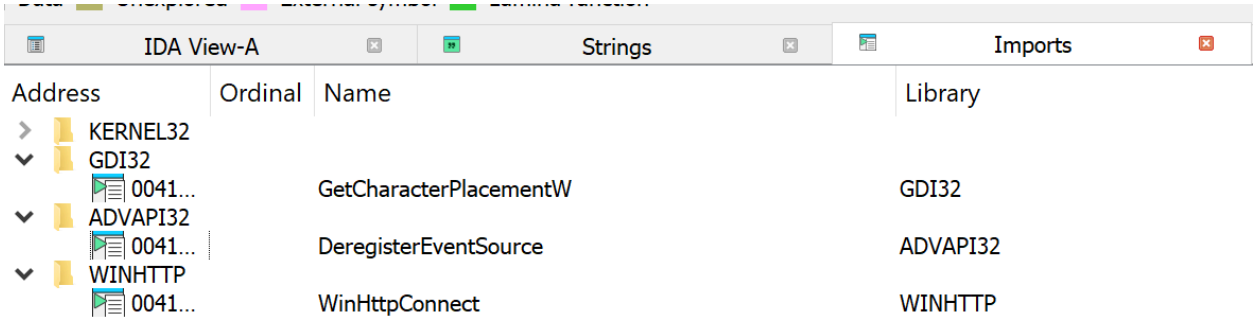
The above screenshot is a zoomed out picture of all of the function calls in this malware. As you can see, it is very detailed and in depth.

Since there were so many function calls, I decided to look into each function and the code that was in each one, as well as the strings used in this malware and the imports used as well.

IDA View-A | Strings

Address	Length	Type	String
.rdata:00411...	00000010	C	no_buffer_space
.rdata:00411...	00000013	C	no_protocol_option
.rdata:00411...	0000000E	C	not_connected
.rdata:00411...	0000000D	C	not_a_socket
.rdata:00411...	00000018	C	operation_not_supported
.rdata:00411...	00000017	C	protocol_not_supported
.rdata:00411...	00000014	C	wrong_protocol_type
.rdata:00411...	0000000A	C	timed_out
.rdata:00411...	00000016	C	operation_would_block
.rdata:00411...	0000001D	C	address family not supported
.rdata:00411...	0000000F	C	address in use
.rdata:00411...	00000016	C	address not available
.rdata:00411...	00000012	C	already connected
.rdata:00411...	00000017	C	argument list too long
.rdata:00411...	00000017	C	argument out of domain
.rdata:00411...	0000000C	C	bad address
.rdata:00411...	00000014	C	bad file descriptor
.rdata:00411...	0000000C	C	bad message
.rdata:00411...	0000000C	C	broken pipe
.rdata:00411...	00000013	C	connection aborted
.rdata:00411...	0000001F	C	connection already in progress
.rdata:00411...	00000013	C	connection refused
.rdata:00411...	00000011	C	connection reset
.rdata:00411...	0000001D	C	destination address required
.rdata:00411...	00000018	C	executable format error
.rdata:00411...	0000000F	C	file too large
.rdata:00411...	00000011	C	host unreachable
.rdata:00411...	00000013	C	identifier removed
.rdata:00411...	00000016	C	illegal byte sequence
.rdata:00411...	00000023	C	inappropriate io control operation
.rdata:00411...	0000000D	C	invalid seek
.rdata:00411...	0000000F	C	is a directory
.rdata:00411...	0000000D	C	message size
.rdata:00411...	0000000D	C	network down
.rdata:00411...	0000000E	C	network reset
.rdata:00411...	00000014	C	network unreachable
.rdata:00411...	00000010	C	no buffer space
.rdata:00411...	00000011	C	no child process
.rdata:00411...	00000008	C	no link
.rdata:00411...	00000015	C	no message available
.rdata:00411...	0000000B	C	no message
.rdata:00411...	00000013	C	no protocol option
.rdata:00411...	00000014	C	no stream resources
.rdata:00411...	0000001A	C	no such device or address

The above screenshot is just some of the strings used in this malware. Looking through these strings, I see some of the same strings from when I used the Strings tool as well as other tools. These strings along with their address helps me understand what each section of the malware is doing, and what certain functions are doing as well.



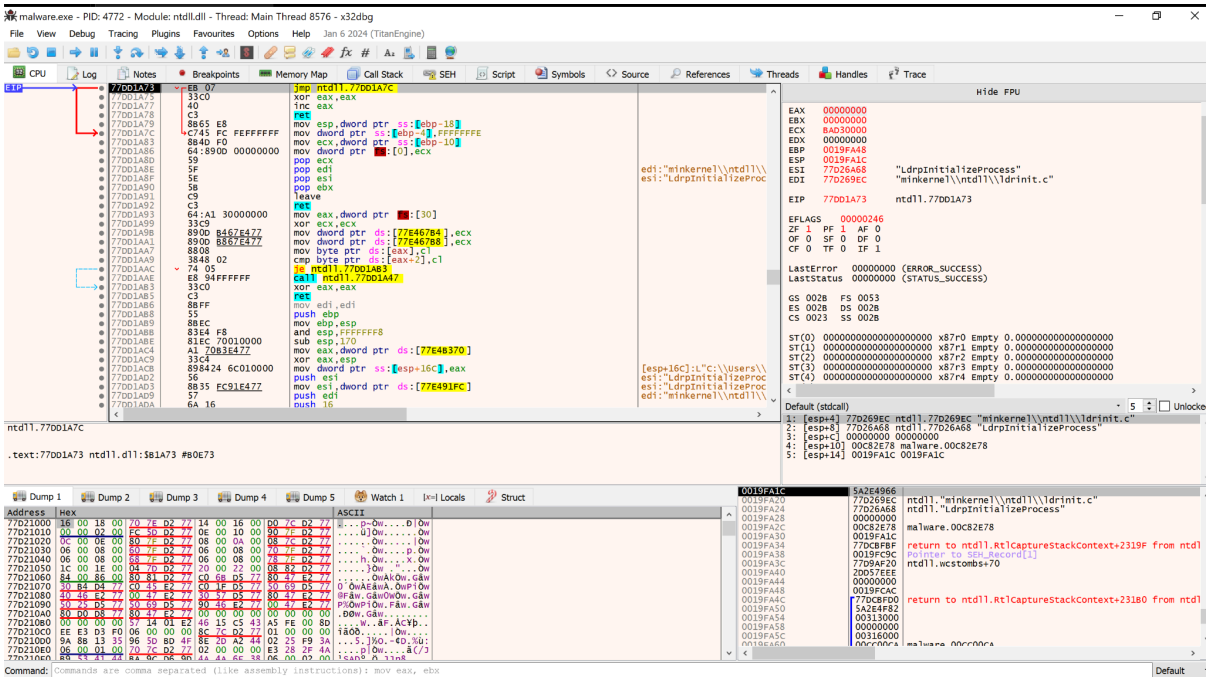
The above screenshot is a list of the imports used in Lumma Stealer. It shows me the DLL's that are imported as well as what functions are used from each DLL. Looking at the functions and their address gives me an understanding of certain roles of each section of this malware.

As I was going through IDA and getting deeper into functions, I realized that the functions listed by IDA for Lumma Stealer on the side panel are mainly for memory allocation and locking, exiting, or aborting. To me this means that this malware can detect when it is being reversed. Since it can detect when it is being reversed, the malware can possibly delete itself or hide information. The rest of the functions that are used in this malware are being called from the DLL imports.

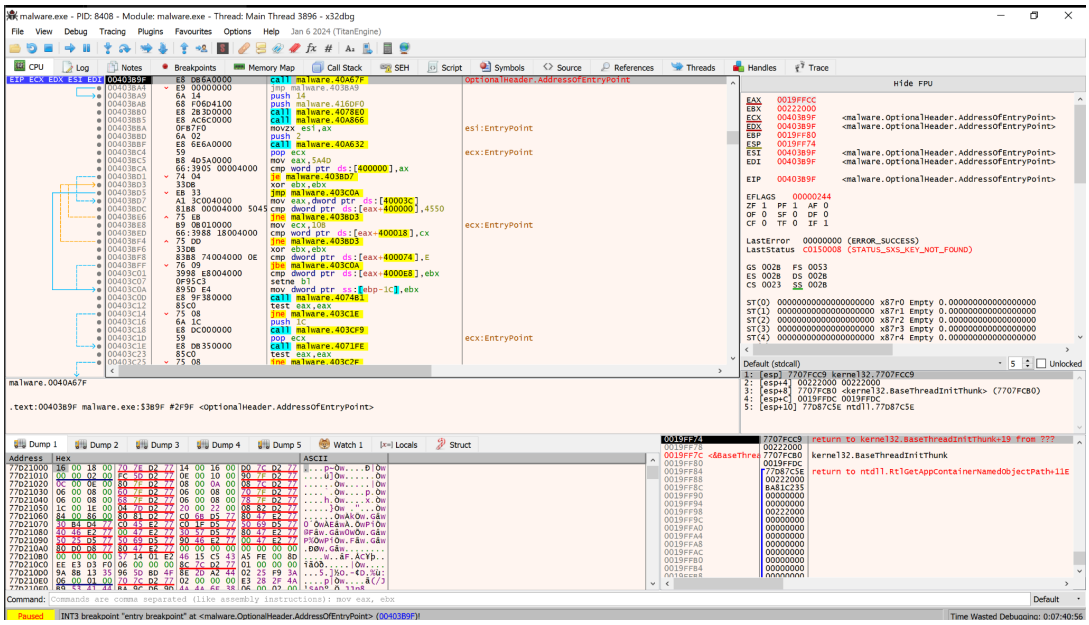
After doing a deep dive into IDA, I obtained a lot of new information about Lumma Stealer. I found out how this malware works, where specific functions are called, and what they do. This malware is able to change notifications and error messages. It also outputs errors and notifications depending on what is happening when it is run. Lumma Stealer can also detect the locale of where it is being run. Once it does this, the malware is able to customize locale specific settings. These include language, date, time, and number formats. This allows the malware to be run anywhere in the world since it can detect the locale of where it is and customize the settings depending on the locale. To go into even more depth, Lumma Stealer can obtain the OEM code page identifier which will allow it to determine the character encoding used by legacy applications. This malware can also do much more. This includes freeing space in memory, specifically the heap. This is done by removing unused blocks of memory. Finally, for the main role of this malware, it can connect to web browsers and obtain information regarding that browser as well as your personal information. Those points stated above are the main functions of Lumma Stealer and how it works. I obtained all of this information by going through IDA and looking into different blocks of code. For any function calls that I was unsure of, I did research on what those functions did. Going through IDA also allowed me to find specific locations for certain functions that I will use in x64dbg and OllyDbg.

Advanced Dynamic Analysis:

x64dbg:



The above screenshot shows the malware I am using opened up in x32dbg. I first tried to open this malware in x64dbg, however, nothing was shown. I tried again with x32dbg and it worked perfectly fine. At first glance, I am able to see the code being run, information regarding the registers, the stack, and the heap. I noticed that when I run the debugger a couple times that it freezes and I can't do anything with the code. I believe this is the code realizing that I am attempting to reverse it.



The above screenshot shows what the output of x32dbg is after I run it once. As you can see the comment next to the top line of code says “OptionalHeader.AddressOfEntryPoint”. This shows that the line of code next to the comment contains additional information about the executable, where it is loaded into memory, and where execution begins.

In order to get to specific areas in the code, I decided to use addresses I found from IDA, as well as lookup string references and use those references to put me in the correct place. Below is a screenshot of some string references found in the malware executable.

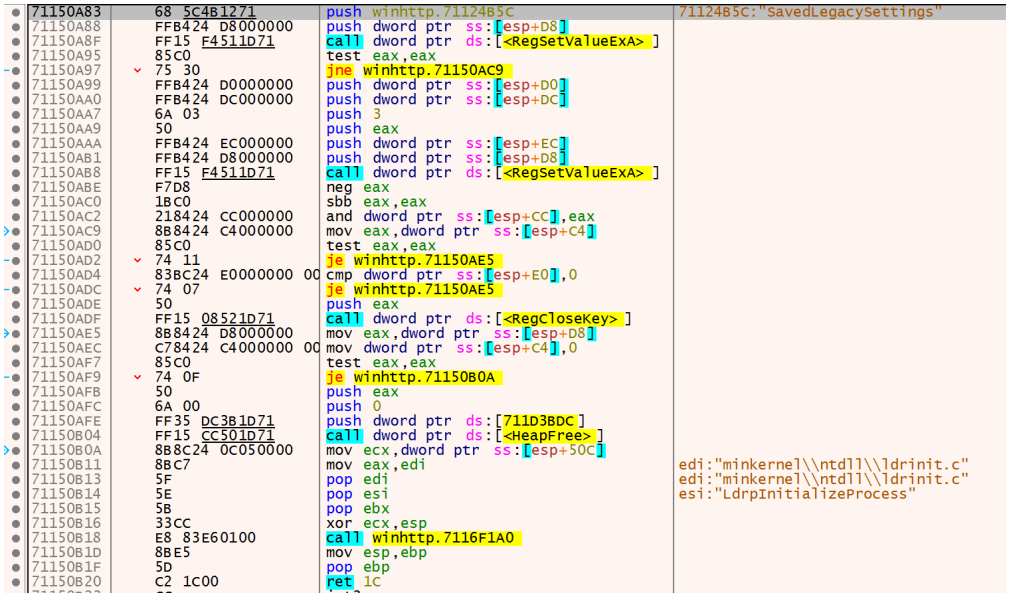
Strings (malware.exe)

Address	Disassembly	String Address	String
004079E9	cmp dword ptr ds:[41208C],0	0041208C	"fc@"
004079F2	push malware.41208C	0041208C	"fc@"
00407C6B	cmp ecx,malware.418B90	00418B90	&"(null)"
00407FAE	movsx eax,byte ptr ds:[eax+412468]	00412468	"rocess"
00408339	mov esi,dword ptr ds:[418B94]	00418B94	&L"(null)"
00408426	mov esi,dword ptr ds:[418B90]	00418B90	&"(null)"
0040878C	mov esi,dword ptr ds:[418B90]	00418B90	&"(null)"
0040BD40	lea ecx,dword ptr ds:[418BA0]	00418BA0	"sqrt"
0040BD9D	lea ecx,dword ptr ds:[418BA0]	00418BA0	"sqrt"
0040A46C	mov eax,dword ptr ds:[eax*8+412654]	00412654	&L"R6002\r\n- floating point support not loaded\r\n"
0040A4D5	push malware.412FF0	00412FF0	L"Runtime Error!\n\nProgram: "
0040A516	push malware.413024	00413024	L"<program name unknown>"
0040A54E	push malware.413054	00413054	L"..."
0040A577	push malware.41305C	0041305C	L"\n\n"
0040A5AF	push malware.413068	00413068	L"Microsoft Visual C++ Runtime Library"
0040A8F9	push malware.4130B4	004130B4	L"kernel32.dll"
0040A90C	push malware.4130D0	004130D0	"FlsAlloc"
0040A91A	push malware.4130DC	004130DC	"FlsFree"
0040A92D	push malware.4130E4	004130E4	"FlsGetValue"
0040A940	push malware.4130F0	004130F0	"FlsSetValue"
0040A953	push malware.4130FC	004130FC	"InitializeCriticalSectionEx"
0040A966	push malware.413118	00413118	"CreateEventExW"
0040A979	push malware.413128	00413128	"CreateSemaphoreExW"
0040A98C	push malware.41313C	0041313C	"SetThreadStackGuarantee"
0040A99F	push malware.413154	00413154	"CreateThreadpoolTimer"
0040A9B2	push malware.41316C	0041316C	"SetThreadpoolTimer"
0040A9C5	push malware.413180	00413180	"WaitForThreadpoolTimerCallbacks"
0040A9D8	push malware.4131A0	004131A0	"CloseThreadpoolTimer"
0040A9EB	push malware.4131B8	004131B8	"CreateThreadpoolWait"
0040A9FE	push malware.4131D0	004131D0	"SetThreadpoolWait"
0040AA11	push malware.4131E4	004131E4	"CloseThreadpoolWait"
0040AA29	push malware.4131F8	004131F8	"FlushProcessWriteBuffers"
0040AA37	push malware.413214	00413214	"FreeLibraryWhenCallbackReturns"
0040AA4A	push malware.413234	00413234	"GetCurrentProcessorNumber"
0040AA5D	push malware.413250	00413250	"GetLogicalProcessorInformation"
0040AA70	push malware.413270	00413270	"CreateSymbolicLinkW"
0040AA83	push malware.413284	00413284	"SetDefaultDllDirectories"
0040AA96	push malware.4132A0	004132A0	"EnumSystemLocalesEx"
0040AAA9	push malware.4132B4	004132B4	"CompareStringEx"
0040AABC	push malware.4132C4	004132C4	"GetDateFormatEx"
0040AACF	push malware.4132D4	004132D4	"GetLocaleInfoEx"
0040AAE2	push malware.4132E4	004132E4	"GetTimeFormatEx"
0040AAF5	push malware.4132F4	004132F4	"GetUserDefaultLocaleName"
0040AB08	push malware.413310	00413310	"IsValidLocaleName"
0040AB1B	push malware.413324	00413324	"LCMapStringEx"
0040AB2E	push malware.413334	00413334	"GetCurrentPackageId"
0040AB41	push malware.413348	00413348	"GetTickCount64"
0040AB59	push malware.413358	00413358	"GetFileInformationByHandleExW"
0040AB67	push malware.413378	00413378	"SetFileInformationByHandleW"
0040B31A	push malware.4133A0	004133A0	"e+000"
004DD27E	push dword ptr ds:[esi*8+414330]	00414330	"hLA"
0040E5DB	push malware.416454	00416454	L"USER32.DLL"
0040E5FD	push malware.416454	00416454	L"USER32.DLL"
0040E612	push malware.41646C	0041646C	"MessageBoxW"

These strings above are only a few of the strings that are referenced in x32dbg.

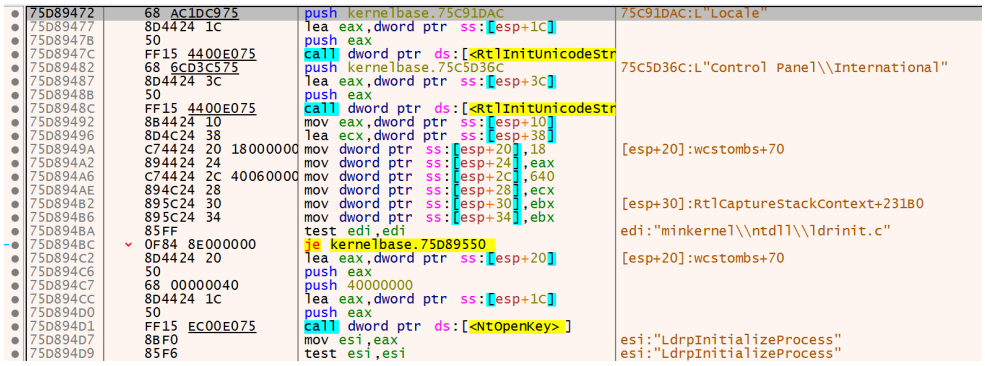
Looking into these strings, and using the addresses found from IDA, I was able to view interesting areas of the code.

I came across a section of the malware that caught my attention. It seems like this section of the malware is what Lumma Stealer is supposed to do. It seems to call the winhttp.dll file and make calls to other functions within this .DLL file. From what I am noticing, this section sets a proxy server, opens a browser, and gathers information from the stack. This is a majority of what Lumma Stealer is supposed to do.

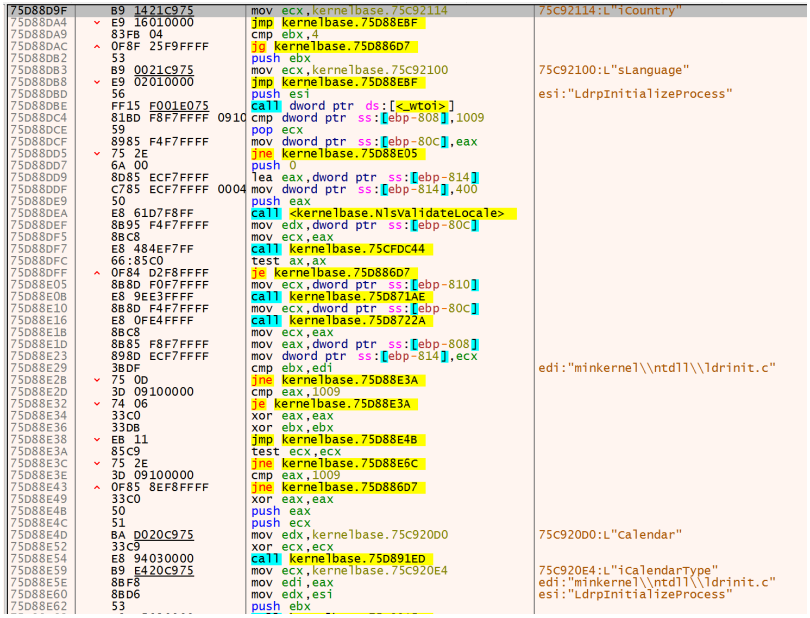


```
71150A83    68 5C4B1271          push winhttp.71124B5C                    71124B5C:"SavedLegacySettings"
71150A88    FFB424 D8000000      push dword ptr ss:[esp+D8]
71150A8F    FF15 F4511D71        call dword ptr ds:[<RegSetValueExA>]
71150A95    85C0                 test eax,eax
71150A97    75 30                jne winhttp.71150AC9
71150A99    FFB424 D0000000      push dword ptr ss:[esp+D0]
71150AA0    FFB424 DC000000      push dword ptr ss:[esp+DC]
71150AA7    6A 03                push 3
71150AA9    50                   push eax
71150AAA    FFB424 EC000000      push dword ptr ss:[esp+EC]
71150AB1    FFB424 D8000000      push dword ptr ss:[esp+D8]
71150AB8    FF15 F4511D71        call dword ptr ds:[<RegSetValueExA>]
71150ABE    F7D8                 neg eax
71150AC0    1BC0                 sbb eax,eax
71150AC2    218424 CC000000      and dword ptr ss:[esp+CC],eax
71150AC9    8B8424 C4000000      mov eax,dword ptr ss:[esp+C4]
71150AD0    85C0                 test eax,eax
71150AD2    74 11                je winhttp.71150AE5
71150AD4    83BC24 E0000000 00   cmp dword ptr ss:[esp+E0],0
71150ADC    74 07                je winhttp.71150AE5
71150ADE    50                   push eax
71150ADF    FF15 08521D71        call dword ptr ds:[<RegCloseKey>]
71150AE5    8B8424 D8000000      mov eax,dword ptr ss:[esp+D8]
71150AEC    C78424 C4000000 00   mov dword ptr ss:[esp+C4],0
71150AF7    85C0                 test eax,eax
71150AF9    74 0F                je winhttp.71150B0A
71150AFB    50                   push eax
71150AFC    6A 00                push 0
71150AFE    FF35 DC3B1D71        push dword ptr ds:[711D3BDC]
71150B04    FF15 CC501D71        call dword ptr ds:[<HeapFree>]
71150B0A    8B8C24 0C050000      mov ecx,dword ptr ss:[esp+50C]
71150B11    8BC7                 mov eax,edi                              edi:"minkernel\\ntdll\\ldrinit.c"
71150B13    5F                   pop edi                                  edi:"minkernel\\ntdll\\ldrinit.c"
71150B14    5E                   pop esi                                  esi:"LdrpInitializeProcess"
71150B15    5B                   pop ebx
71150B16    33CC                 xor ecx,esp
71150B18    E8 83E60100          call winhttp.7116F1A0
71150B1D    8BE5                 mov esp,ebp
71150B1F    5D                   pop ebp
71150B20    C2 1C00              ret 1C
```

The screenshot above shows the part of the code where the malware obtains the legacy settings of applications. This is used to determine the type of character encoding.



```
75D89472    68 AC1DC975          push kernelbase.75C91DAC                 75C91DAC:L"Locale"
75D89477    8D4424 1C            lea eax,dword ptr ss:[esp+1C]
75D8947B    50                   push eax
75D8947C    FF15 4400E075        call dword ptr ds:[<RtlInitUnicodeStr
75D89482    68 6CD3C575          push kernelbase.75C5D36C                 75C5D36C:L"Control Panel\\International"
75D89487    8D4424 3C            lea eax,dword ptr ss:[esp+3C]
75D8948B    50                   push eax
75D8948C    FF15 4400E075        call dword ptr ds:[<RtlInitUnicodeStr
75D89492    8B4424 10            mov eax,dword ptr ss:[esp+10]
75D89496    8D4C24 38            lea ecx,dword ptr ss:[esp+38]
75D8949A    C74424 20 18000000   mov dword ptr ss:[esp+20],18             [esp+20]:wcstombs+70
75D894A2    894424 24            mov dword ptr ss:[esp+24],eax
75D894A6    C74424 2C 40060000   mov dword ptr ss:[esp+2C],640
75D894AE    894C24 28            mov dword ptr ss:[esp+28],ecx
75D894B2    895C24 30            mov dword ptr ss:[esp+30],ebx            [esp+30]:RtlCaptureStackContext+231B0
75D894B6    895C24 34            mov dword ptr ss:[esp+34],ebx
75D894BA    85FF                 test edi,edi                             edi:"minkernel\\ntdll\\ldrinit.c"
75D894BC    0F84 8E000000        je kernelbase.75D89550
75D894C2    8D4424 20            lea eax,dword ptr ss:[esp+20]            [esp+20]:wcstombs+70
75D894C6    50                   push eax
75D894C7    68 00000040          push 40000000
75D894CC    8D4424 1C            lea eax,dword ptr ss:[esp+1C]
75D894D0    50                   push eax
75D894D1    FF15 EC00E075        call dword ptr ds:[<NtOpenKey>]
75D894D7    8BF0                 mov esi,eax                              esi:"LdrpInitializeProcess"
75D894D9    85F6                 test esi,esi                             esi:"LdrpInitializeProcess"
```

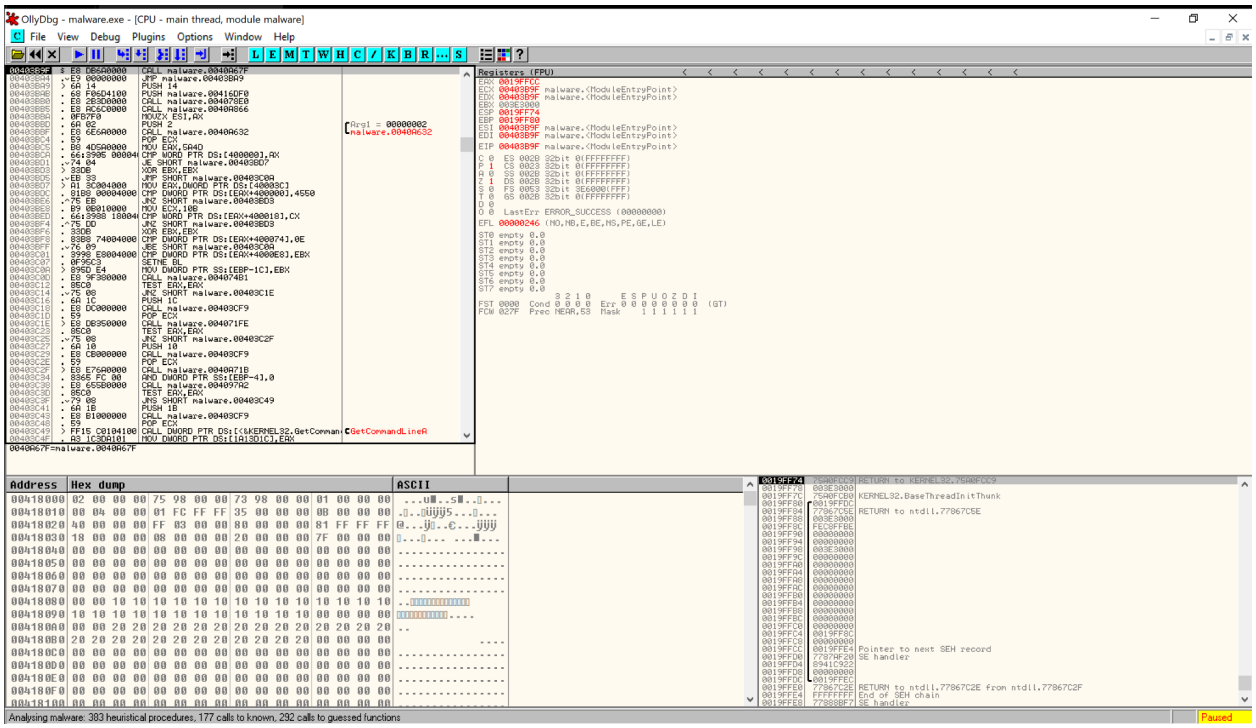


The two screenshots above show sections of code in the malware where the locale is defined based off of the country and language. This allows the malware to alter its settings such as date, time, language, and number format based on where in the world it is being run.

The screenshot above shows samlib.dll being used in this malware to open a domain, free memory, and change passwords of users. This is very interesting and it is the first time I noticed that the malware is doing something like this.

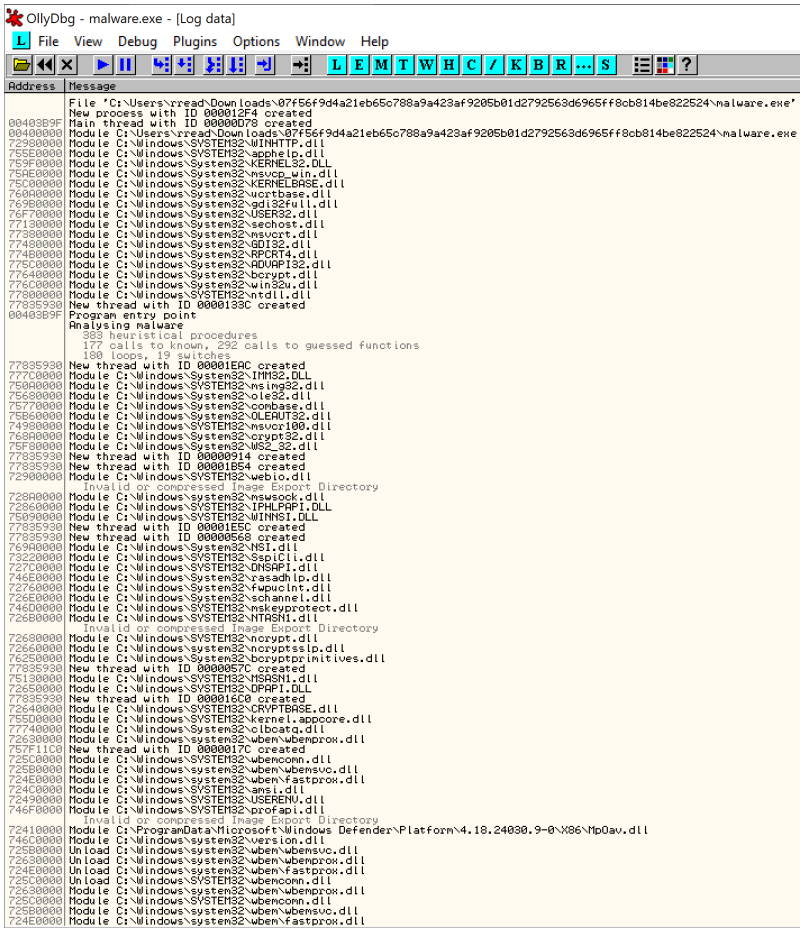
Overall, when I took a deep dive into x32dbg, I noticed a lot of interesting things, some of which I hadn't noticed before. I was able to see what Lumma Stealer did when it found out that it was being reversed. I was also able to see major parts of Lumma Stealer such as code where it created a proxy and connected to a web browser, freeing memory, changing users passwords, identifying the locale and altering its settings based on that, and so much more.

OllyDbg:

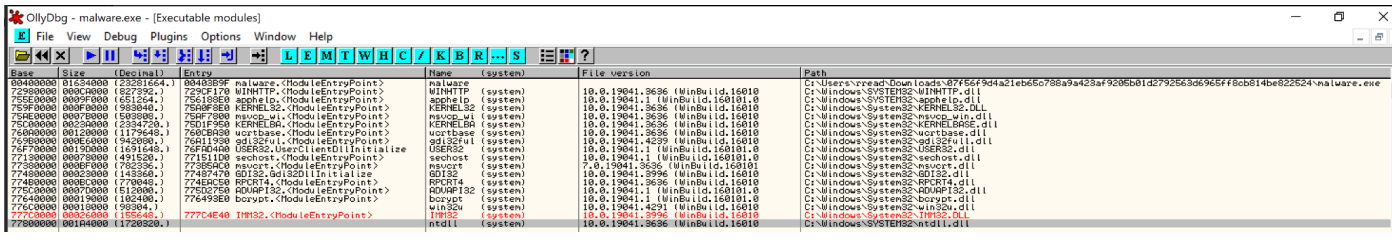


The screenshot above shows the malware executable opened in OllyDbg. Once opened, I am able to see code, register information, information on the stack and heap. When I attempt to run the executable, the screen freezes and I am unable to do anything. This is the same thing that happened when I attempted to run it in x64dbg.

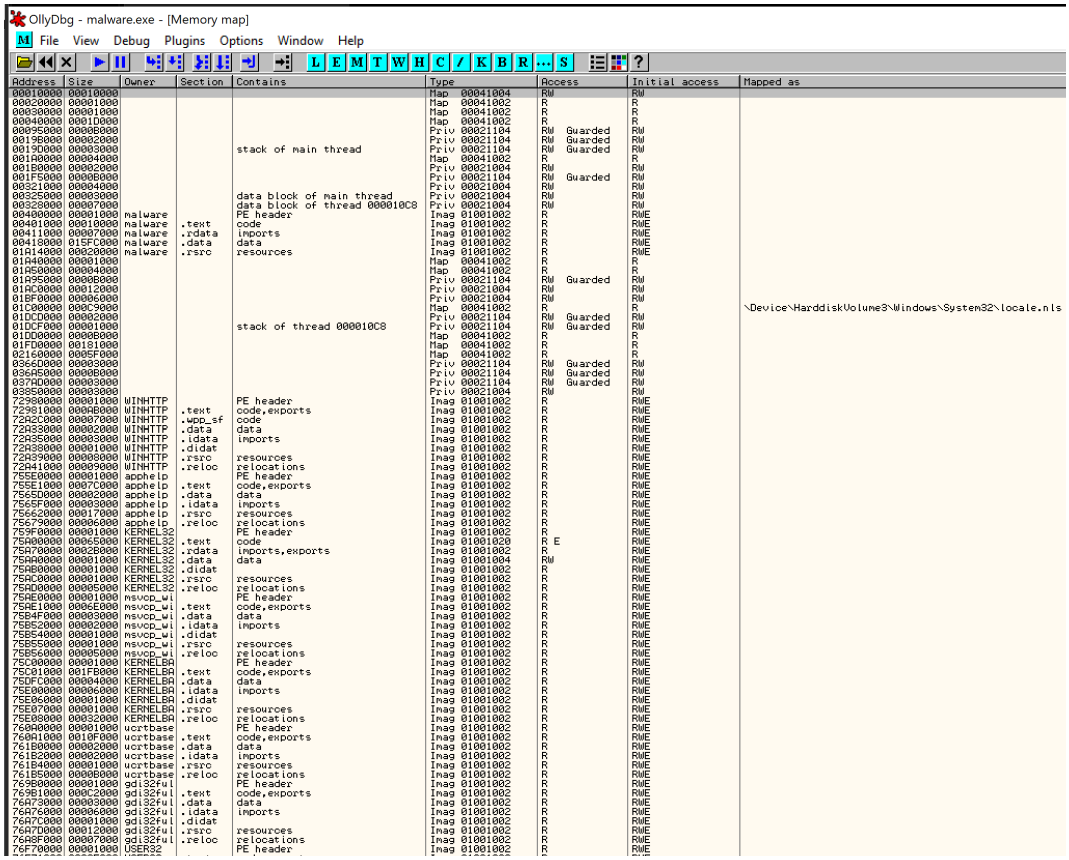
Since OllyDbg and x64dbg are both debuggers and ultimately do the same thing, in OllyDbg I decided to view alternative information. I first started by looking into the log, executable, and memory files.



The above screenshot shows the log file. This file is basically a log of everything that happened when the malware ran. It is very interesting to look into and see the files that were opened and .DLL files that were run. It also allows you to see the process and its ID that were created as well as the threads and their ID's.



The above screenshot shows all of the executable modules that were run. It gives information regarding where in the debugger they were run, and their size. It also gives information about the executable module itself such as the name, version of it, and its path.



The screenshot above shows the memory map of the executed malware. In the memory map, it includes the owners of the .DLL files along with the section and what it contains. It also provides the access, and initial access as well as what it is mapped to.

The three files listed above are all helpful in their own ways.

As I look through the CPU view of OllyDbg, I notice a lot of familiar function calls. I remember seeing a majority of these functions while looking through IDA and x64dbg.

Overall, since OllyDbg and x64dbg are used for the same purpose, I didn't find a lot of new information in OllyDbg. I still went through OllyDbg and used certain features that it has to help me find information. I also found some of the addresses that I was looking into from x64dbg. I think that x64dbg helped me gather more information, was more organized and had more features than OllyDbg. However, OllyDbg is still a good tool to use for reverse engineering if you need a debugger.

Summary:

By utilizing the tools provided in Basic Static, Basic Dynamic, Advanced Static, and Advanced Dynamic Analysis, I was able to learn a lot about Lumma Stealer. These tools allowed me to reverse this malware and figure out what exactly it does and how. After utilizing the reverse engineering tools I was also able to see the potential danger of Lumma Stealer. Lumma Stealer is a dangerous piece of malware that can be downloaded onto your device through phishing or social engineering tactics. Once downloaded, Lumma Stealer works in the background and is ultimately undetectable to a normal user. Lumma Stealer will aim to steal your sensitive information including usernames and passwords, and personal information. Lumma Stealer can also open a web browser and steal important information from the web browser. Once the information is gathered, the threat actors will use a proxy server to exfiltrate the data to. As you can tell this malware can be dangerous since the threat actors now have your personal and sensitive information. In order to protect against Lumma Stealer, you must have an antivirus running on your device. You should also avoid clicking on links or unknown files. You should only click on links or download files if they are from a trusted source. If Lumma Stealer is already downloaded onto your device, it can be tricky to remove. It will attempt to evade detection and removal. However you can attempt to remove it. You can do this by running an antivirus scan on your device to detect malicious files or viruses. You can then manually delete these files from your device. Once this is completed, restart your web browsers and make sure your operating system is updated. In order to make sure these steps worked, you can use trusted malware removal tools.