a. Virtualization phase:

Virtualization was fairly simple. We initially used VMware Workstation 17 Player for both the Windows 10 and Windows XP machines. Windows XP worked just fine, but our Windows 10 machine was very slow and laggy and sometimes crashed, and after looking online it looked like that was a problem a lot of people faced, so we decided to use Oracle VirtualBox for our Windows 10 machine. It was much faster and did not crash.

Note: most tools are used on Windows 10 unless otherwise stated.

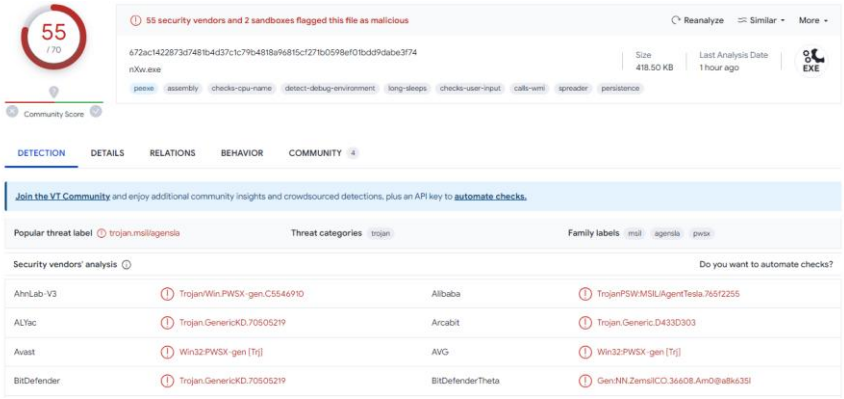
b. Malware collection phase:

We found our malware on https://bazaar.abuse.ch/ (Malware Bazaar). Finding an infected .exe file was very straightforward, many samples were available online.

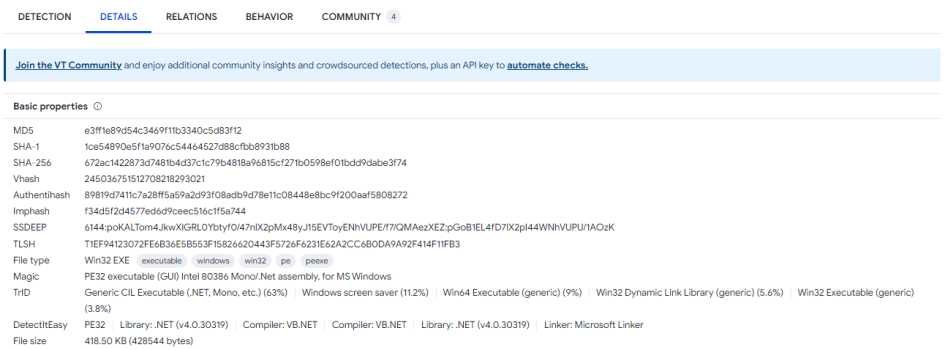
c. Static malware analysis phase:

1. **General analysis:**
   - **VirusTotal:**

Immediately, the file displays extremely suspicious behavior:



As we can see, most malware analysis websites classified the file as malicious. If we take a closer look at the file's details:



Nothing suspicious in basic properties. If we move on to the history of the file, however:

History

Creation Time	2095-12-16 23:44:49 UTC
First Seen In The Wild	2023-11-29 12:32:14 UTC
First Submission	2023-11-21 11:40:42 UTC
Last Submission	2023-11-28 13:54:45 UTC
Last Analysis	2023-11-29 21:00:21 UTC

The creation time of the file is set to be in 2095, this is the first clear sign of malicious activity and file obfuscation.

If we look at the names for the file:

Names

672ac1422873d7481b4d37c1c79b4818a96815cf271b0598ef01bdd9dabe3f74.exe
nXw.exe
njrat_qui_a_casse_extracteur_nsm.bin
purches order.exe
USVrid.exe

“purches order.exe” raises some suspicion, file may be linked to some shopping website, attacker also possibly misspelled purchase to avoid the file being detected.

Digital signature:

Signature info

Signature Verification

⚠ File is not signed

No digital signature, typical for malware to avoid detection, makes it challenging to verify if this file came from a legitimate source.

Sections in the PE info:

Sections

Name	Virtual Address	Virtual Size	Raw Size	Entropy	MD5	Chi2
.text	8192	425800	425984	7.91	4295b0effeb077573655dc55fe3a0af9	136136.12
.rsrc	434176	1452	1536	4.08	f6177b481524abbebafa5ce5d97e91a9	78507.83
.reloc	442368	12	512	0.1	28d762c37b2f106bb87e683103d96ca3	128015

Three things can be found from the screenshot above:

a. Virtual address for .text section is very far from other two sections. In a well-structured and properly compiled executable file, the section addresses are typically contiguous; this address seems suspicious.

b. Entropy in .text section is extremely high, big indicator of malicious activity

c. Very big difference in Virtual vs. Raw size in .reloc section, also indicates malicious behavior

These were the most obvious signs found using VirusTotal.

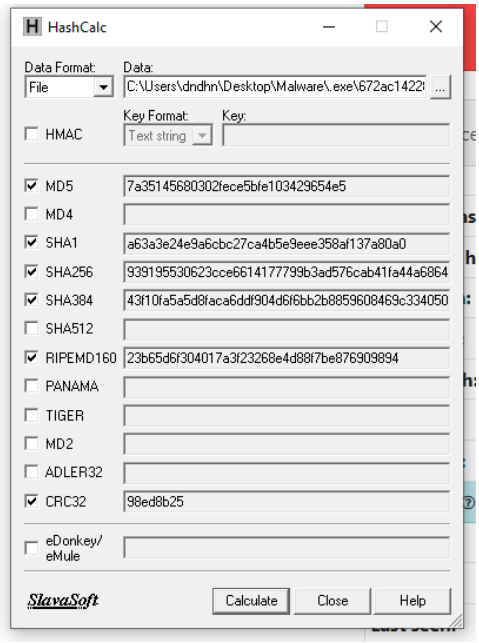
2. Hash calculations:

- **HashCalc:**

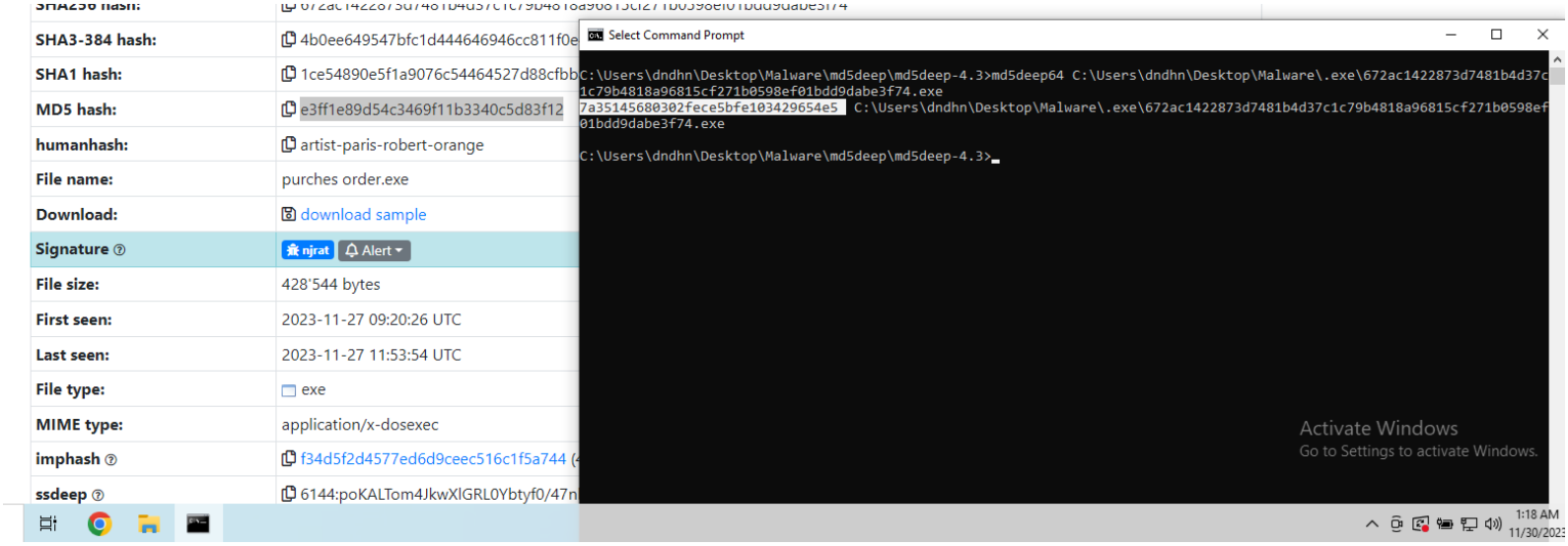
The original hashes provided by the author of the file were as follows:

SHA256 hash:	672ac1422873d7481b4d37c1c79b4818a96815cf271b0598ef01bdd9dabe3f74
SHA3-384 hash:	4b0ee649547bfc1d444646946cc811f0e4ffb7d4e5fa7214ab1a707ee9bb837ba9ea9418e1f8b20cc4e2be2cbd3faab3
SHA1 hash:	1ce54890e5f1a9076c54464527d88cfbb8931b88
MD5 hash:	e3ff1e89d54c3469f11b3340c5d83f12
humanhash:	artist-paris-robert-orange
File name:	purches order.exe
Download:	download sample
Signature	njrat Alert

The hashes provided by the HashCalc tool, however, did not match the original checksums:



- **MD5Deep:**



In the screenshot above, we can see the original hash (*left*) and the resulting hash from the MD5Deep tool (*right)* also don't match.

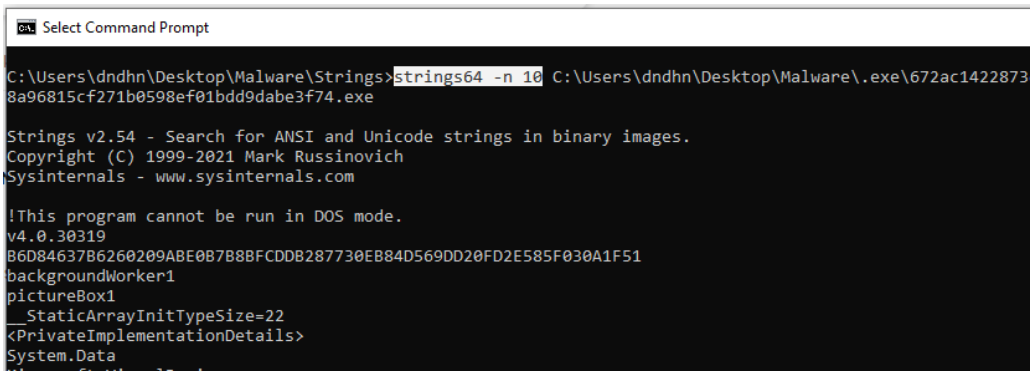
Although there might be other reasons for hash differences, such as file corruption or a software patch, it could also be a sign of file tampering and obfuscation.

Comparing hashes is a great start to detecting malicious activity, but it's not enough on its own to prove it. We need additional information to determine whether or not the file is really infected.

3. Strings and binary analysis:

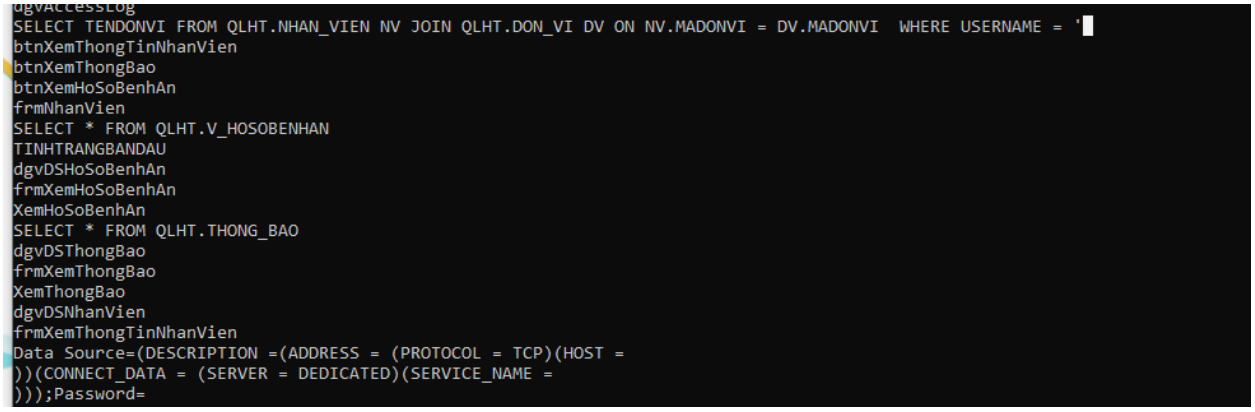
- **Strings:**

Using the strings command, our file appeared to have a very large number of strings, most of which were pretty insignificant to our analysis. We limited the length of the strings shown to 10 or more characters, in hopes of getting some human-readable text:



Some findings:

- The file contained many SQL commands, mostly SELECT statements, which indicates that the file might be a script related to some database.

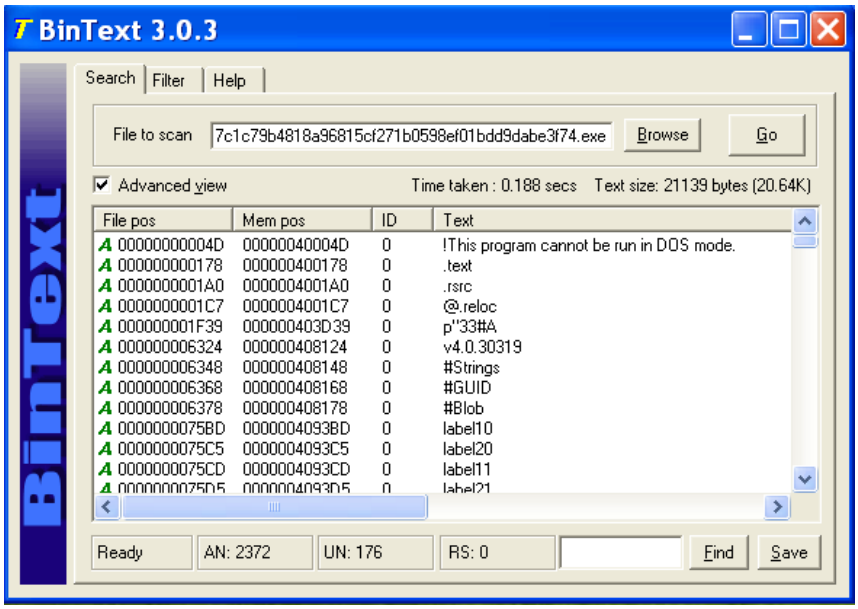


- The words "*Role*", "*Privellege*" and "*Password*" appeared multiple times in the file, which suggests the file may be responsible for access control and security configuration. This could be related to defining user roles and their associated privileges.

- .NET framework v4 is used for the application. mscore.dll file is also present, which is a critical component for managing the execution of .NET applications. It helps handle tasks such as memory management, exception handling, and the loading and execution of managed code. Some copyright information can also be seen.

Our file seems to be some kind of role and privilege manager, related to a database, which can be seriously dangerous if it is indeed obfuscated. The strings tool helped us learn a lot about the file, but nothing definitively malicious was found using the tool. While some suspicious text, and a lot of potentially encrypted information was found, it's important to remember that these patterns can also exist in typical .exe files, so they are not enough to conclude that a file is malicious, but they are a good start to analyzing a file.

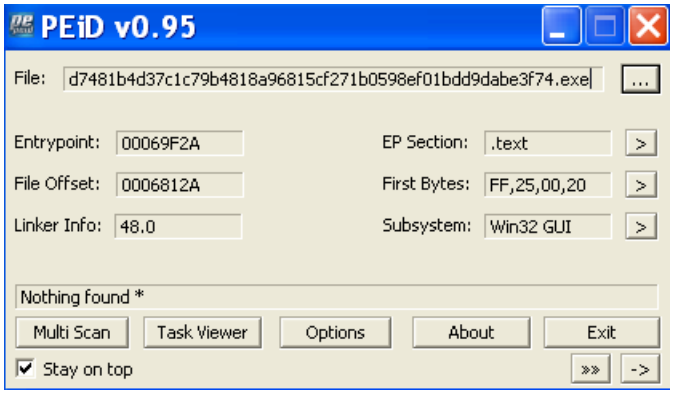
- **BinText:** *(Windows **XP**)*



Same outputs as the strings tool, but some things made BinText easier:

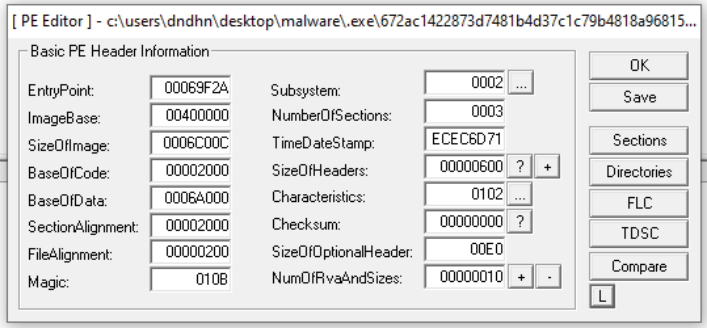
- You can see text, function calls, IP addresses.
- You can also filter the way things are viewed using filter button, in a much easier way. You can modify:
  - What characters to include/ not to include in the definition of strings
  - String size (min./Max. length)
  - Essentials

4. **Portable Executable file viewers:**
   - **PEiD:** *(Windows **XP**)*

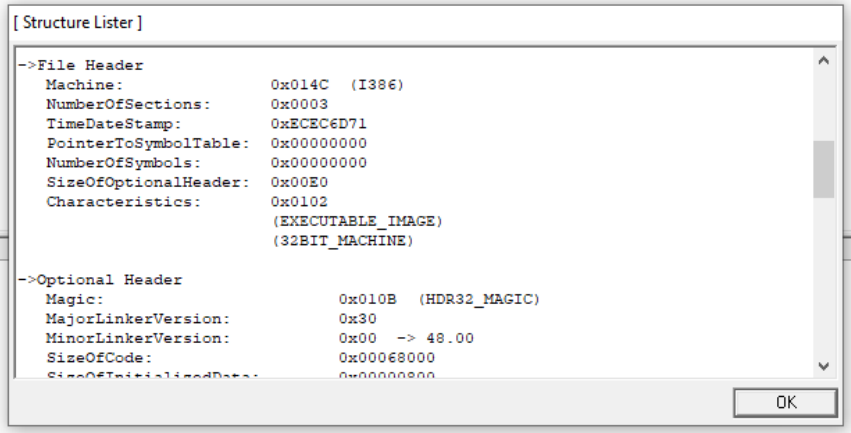


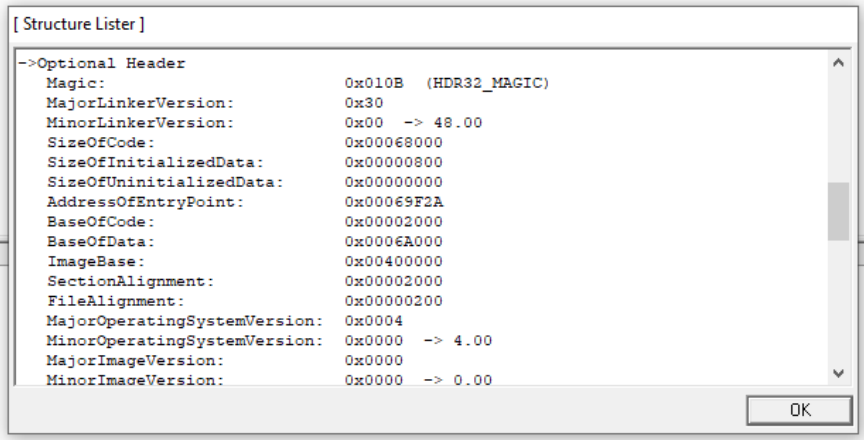
Not a lot of information about the file, however, a good thing about PEiD is that it gives the ability to see the offset and task view. In this malware, when detecting the packers, nothing was found (no files are packed) according to PEiD. It also tells us it is GUI program.

- **LordPE**:



List of some important fields:



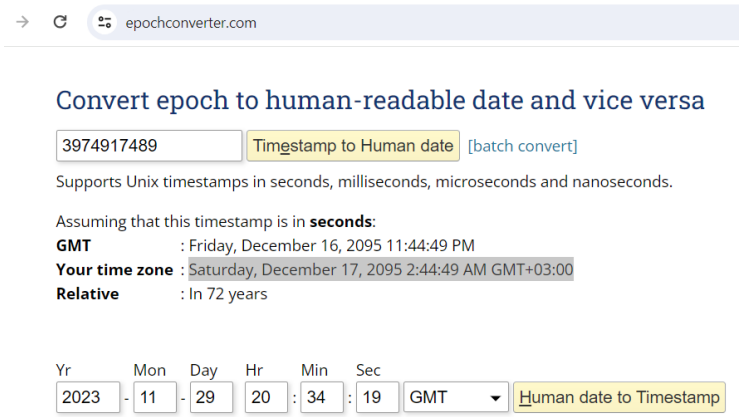


After referring to **PE file format**, to compare the values above to the values you would typically get from an executable (image) file, we concluded the following:

Most of the fields' values are valid:

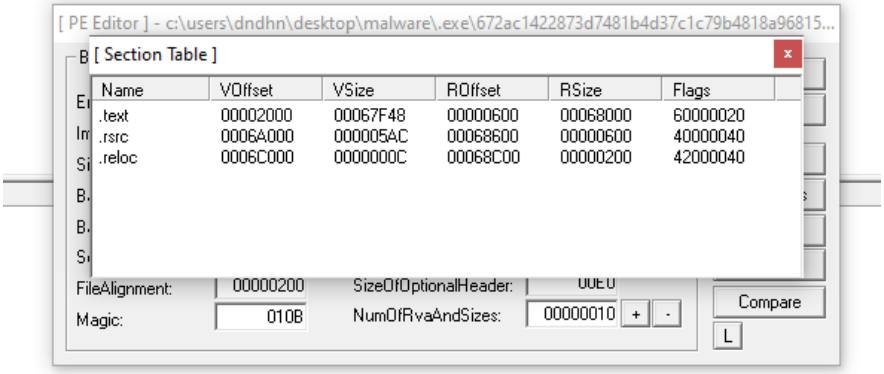
- ***EntryPoint*** should fall in the range between (ImageBase – ImageBase+SizeOfCode); 0x00069F2A <= (0x00400000 + 0x00068000), so EntryPoint is valid. It's important to look at what this location points to, however.
- ***ImageBase*** (0x00400000) is the default value for Windows 10: also valid.
- ***SectionAlignmnet*** should be greater than ***FileAlignment***: also both valid.
- ***NumberOfSections*** is 3, for the .text, .rsrs and .reloc sections: valid.

However, after converting ***TimeDateStamp*** field to a human-readable date, we get the following:



As we can see in the screenshot above, the supposed time of creation for the file is in the year 2095, this is possibly the biggest sign that there was an attempt to manipulate or obfuscate information.

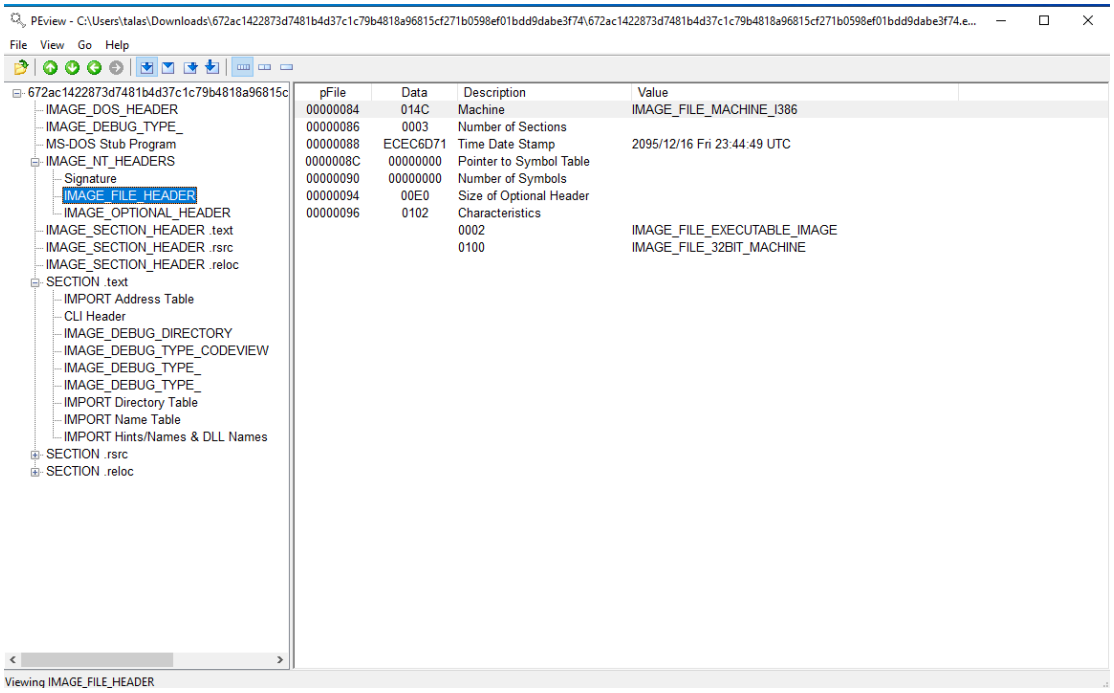
Lastly, if we take a closer look at the Sections in the file:



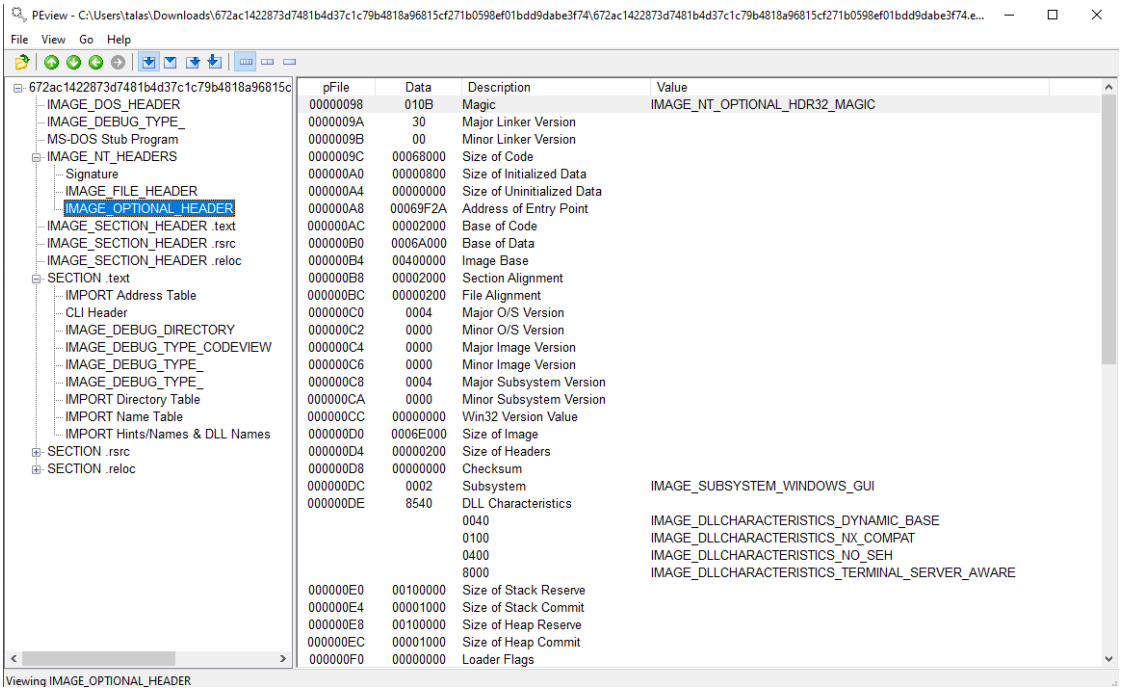
The .rsrc section seems fine, but the **.text** and **.reloc** sections display suspicious behavior: the *.text* section's address is nowhere near the other two sections, and as we mentioned before, these sections are typically contiguous which makes this address suspicious. *.reloc* section: typically, the VSize is equal to or larger than the RSize for sections, especially for the .reloc section, which often involves memory adjustments during runtime. This is also a sign of malicious tampering of the file, possibly obfuscating the information in the .reloc section.

- **PEview:**

A cleaner look at our findings:

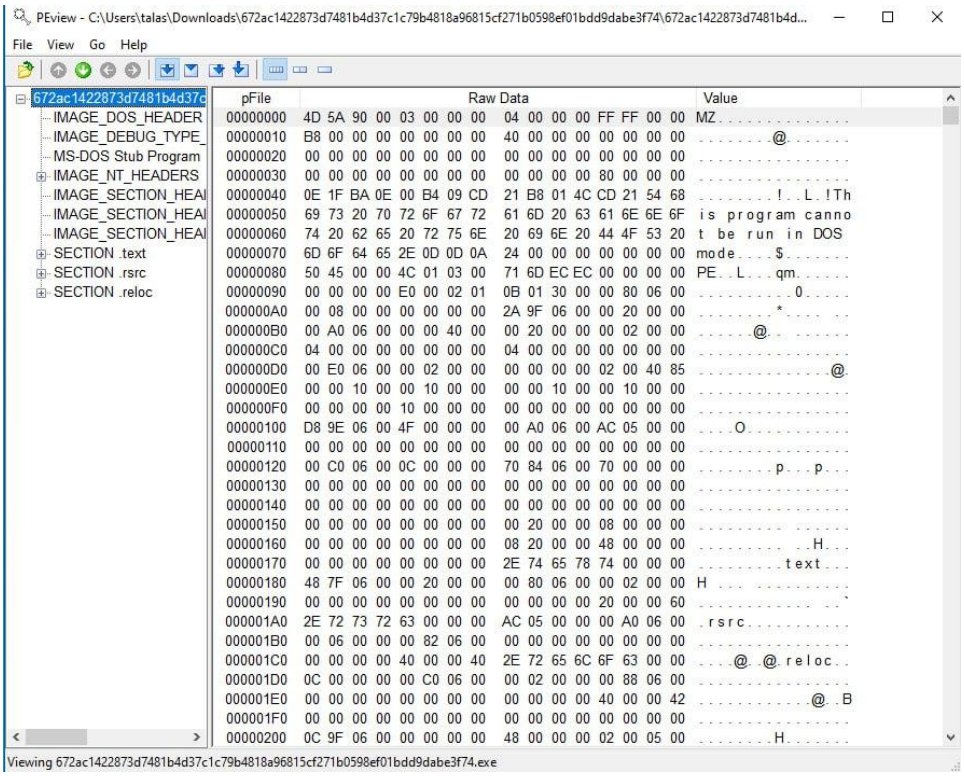


As we mentioned previously, the Time Date Stamp is the most obvious indicator that this file was tampered with. PEview makes it a lot easier to detect that as it translates the values for us.

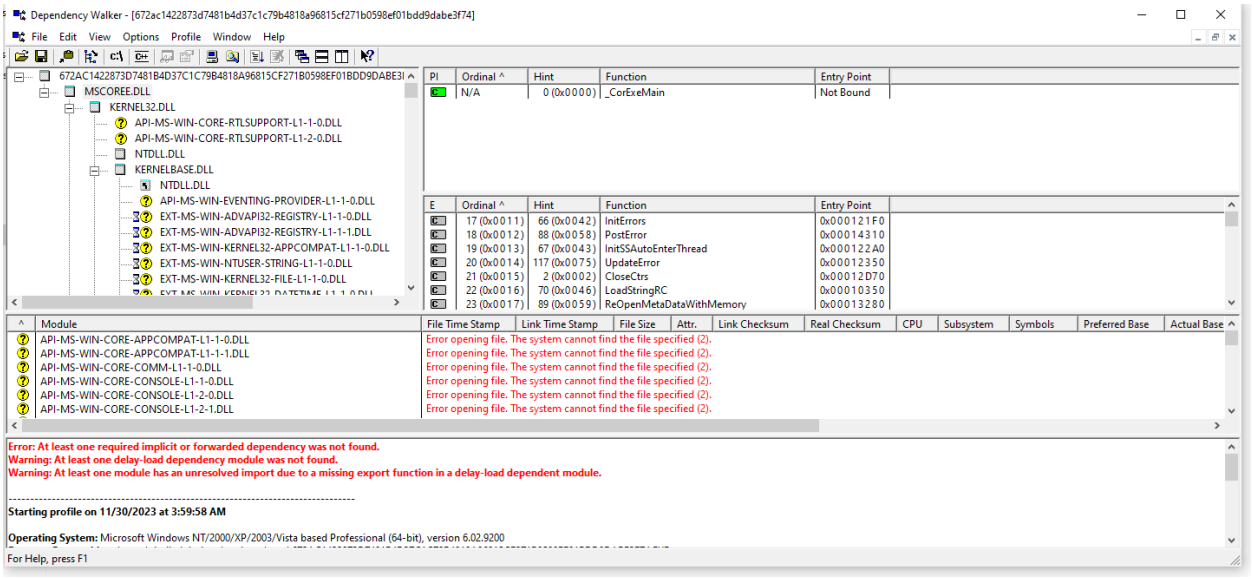


All the information above also matches the information gathered from LordPE, nothing suspicious about the fields here, all values seem to fall within a valid range.

Hex view:


PEview - C:\Users\talas\Downloads\672ac1422873d7481b4d37c1c79b4818a96815cf271b0598ef01bdd9dabe3f74\672ac1422873d7481b4d...
File View Go Help
672ac1422873d7481b4d37c
IMAGE_DOS_HEADER
IMAGE_DEBUG_TYPE_
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEAI
IMAGE_SECTION_HEAI
IMAGE_SECTION_HEAI
SECTION .text
SECTION .rsrc
SECTION .reloc
pFile Raw Data Value
00000000 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ
00000010 B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 @
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00
00000040 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 !..L.!Th
00000050 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00000060 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS
00000070 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 mode.$
00000080 50 45 00 00 4C 01 03 00 71 6D EC EC 00 00 00 00 PE..L...qm
00000090 00 00 00 00 E0 00 02 01 0B 01 30 00 00 80 06 00 0
000000A0 00 08 00 00 00 00 00 00 2A 9F 06 00 00 20 00 00 *
000000B0 00 A0 06 00 00 00 40 00 00 20 00 00 00 02 00 00 @
000000C0 04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00
000000D0 00 E0 06 00 00 02 00 00 00 00 00 00 02 00 40 85 @
000000E0 00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00
000000F0 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00
00000100 D8 9E 06 00 4F 00 00 00 00 A0 06 00 AC 05 00 00 O
00000110 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000120 00 C0 06 00 0C 00 00 00 70 84 06 00 70 00 00 00 p...p...
00000130 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000150 00 00 00 00 00 00 00 00 00 20 00 00 08 00 00 00
00000160 00 00 00 00 00 00 00 00 08 20 00 00 48 00 00 00 H
00000170 00 00 00 00 00 00 00 00 2E 74 65 78 74 00 00 00 text
00000180 48 7F 06 00 00 20 00 00 00 80 06 00 00 02 00 00 H
00000190 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00 60 `
000001A0 2E 72 73 72 63 00 00 00 AC 05 00 00 00 A0 06 00 .rsrc
000001B0 00 06 00 00 00 82 06 00 00 00 00 00 00 00 00 00
000001C0 00 00 00 00 40 00 00 40 2E 72 65 6C 6F 63 00 00 @..@.reloc
000001D0 0C 00 00 00 00 C0 06 00 00 02 00 00 00 88 06 00
000001E0 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 42 @..B
000001F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000200 0C 9F 06 00 00 00 00 00 48 00 00 00 02 00 05 00 H
Viewing 672ac1422873d7481b4d37c1c79b4818a96815cf271b0598ef01bdd9dabe3f74.exe

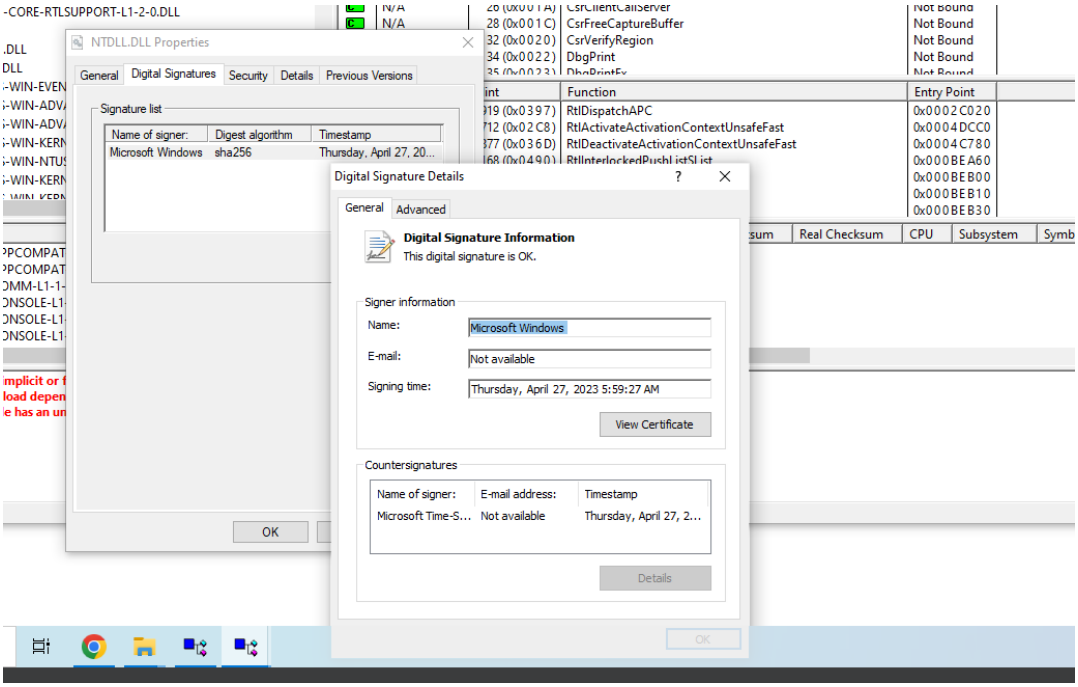
- **Dependency Walker:**



***note: these errors are just compatibility errors, they are not indicators of malicious activity.*

As shown above, the file uses a lot of .dll modules, mainly *mscoree.dll*: a critical component of the CLR (Microsoft Common Language Runtime), which is responsible for managing the execution of .NET applications, and *kernel32.dll*.

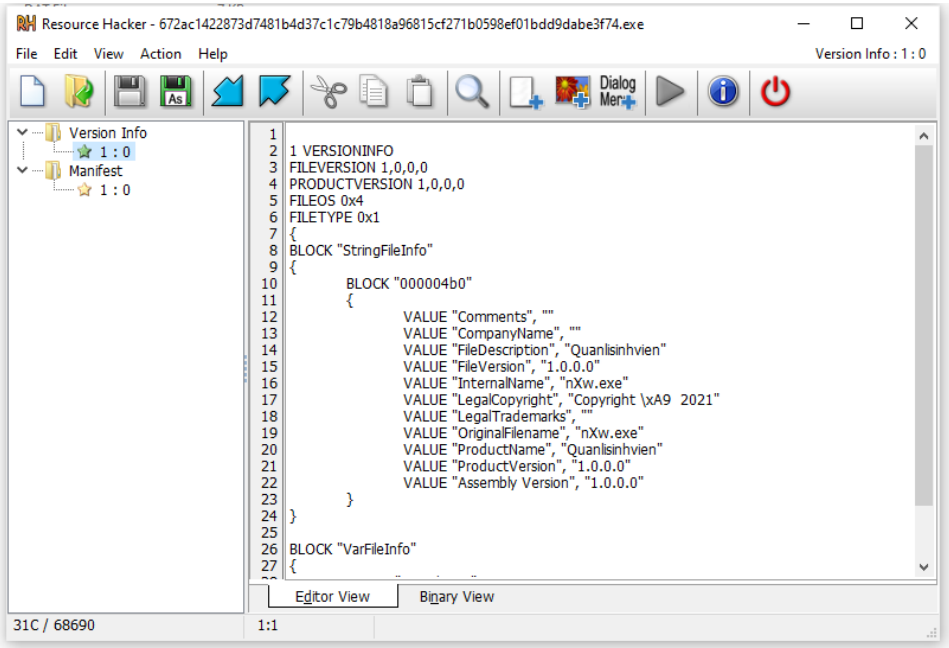
We used dependency walker to check for unusual behavior in these modules, typically when a module is verified it will give us something like this (*NTDLL.DLL*):



However, this signature was not available in a lot of modules, a lot were unsigned which indicates that the file is illegitimate.

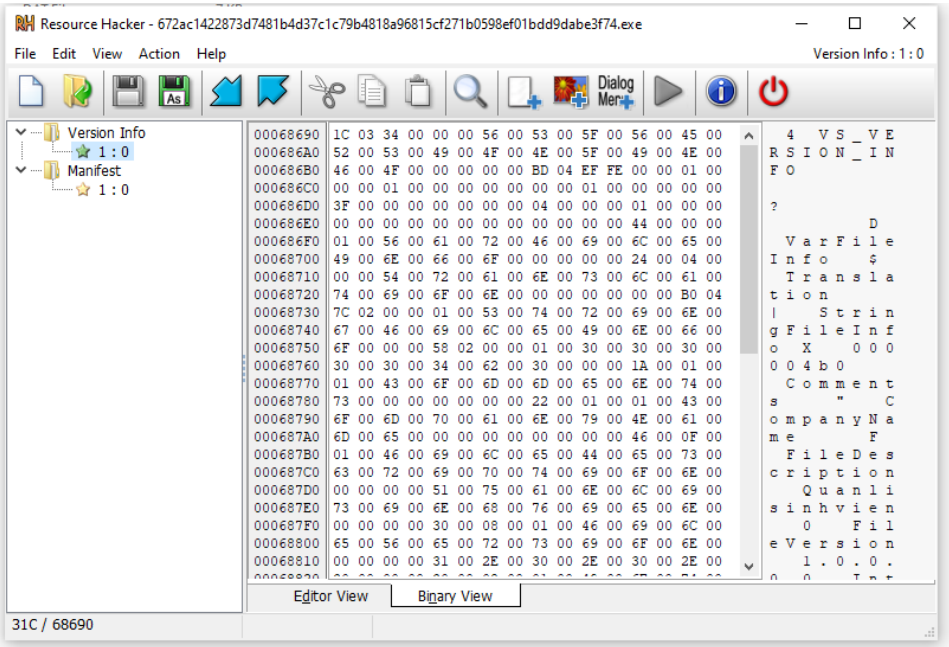
5. Resource analysis:

- **Resource Hacker:**



Version info seems to be standard, the only suspicious thing concluded here is the FileDescription: Quanlisinhvien, which after some searching it seems it's not a real word, so it could be an indicator but still, information provided about this file is very vague. Manifest file also contained nothing particularly suspicious.

Hexadecimal values:

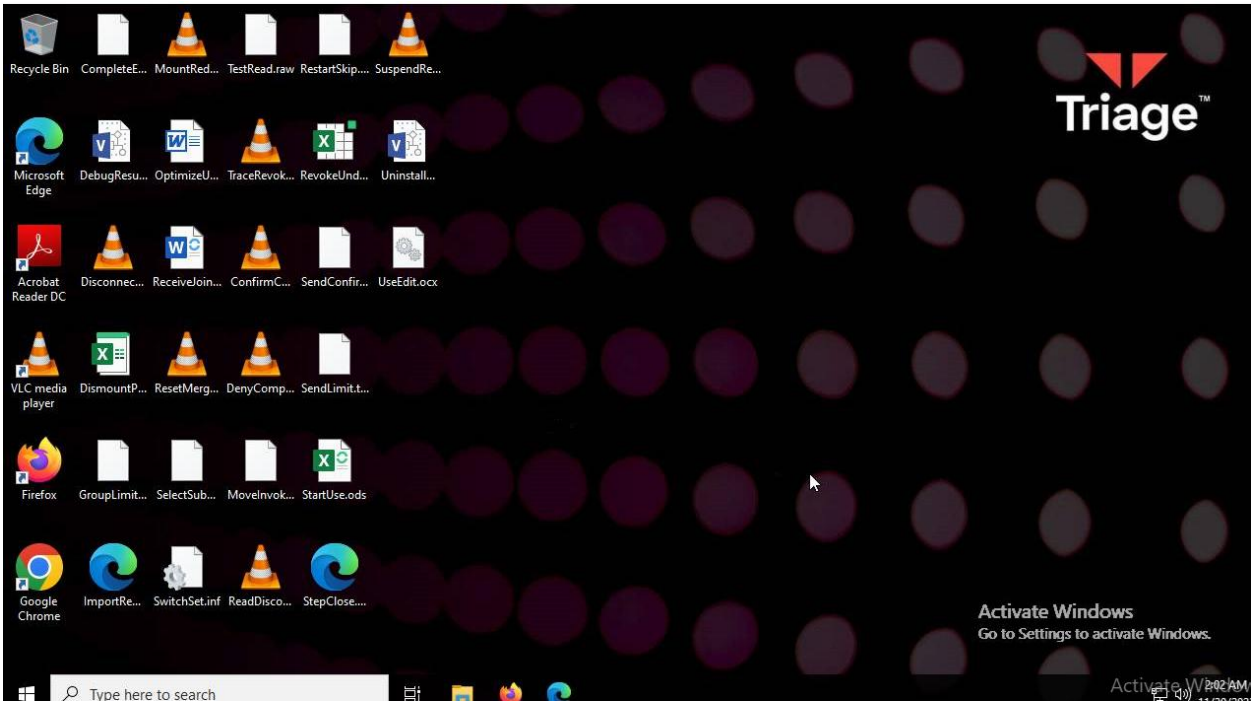


d. Dynamic analysis phase:

- ### Free sandbox tool:

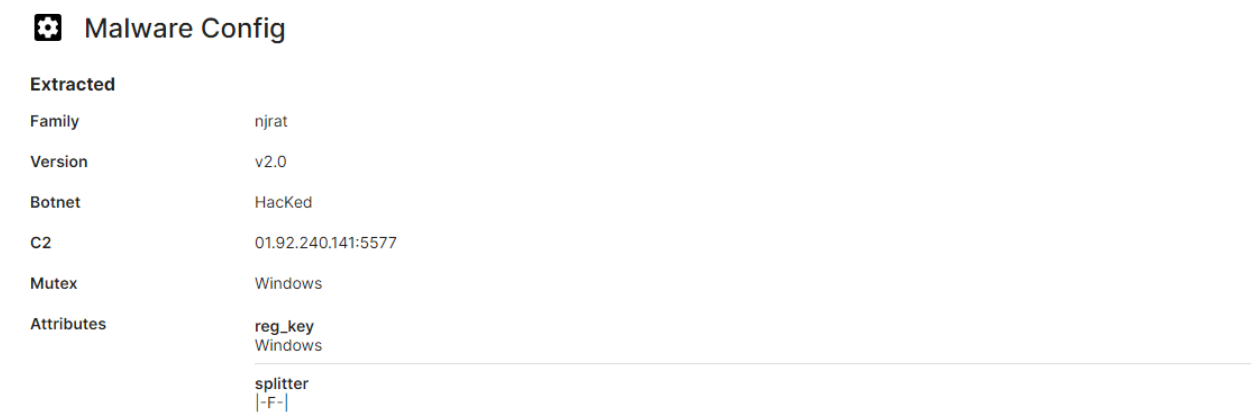
Unfortunately, free versions of sandboxing tools are very limited in what they offer. We tried to use *Any.Run* but the only free operating system it supported was Windows 7. *JoeSandbox* also did not approve our account.

We ended up using Recorded Future's Triage sandbox tool (https://tria.ge/). This tool basically opens the executable and gives a full report of its behavior in the sandbox, any changes occurring after executing, what kind of malicious content the file contains… etc. This is what the sandbox looks like (we chose Windows 10 64bit):

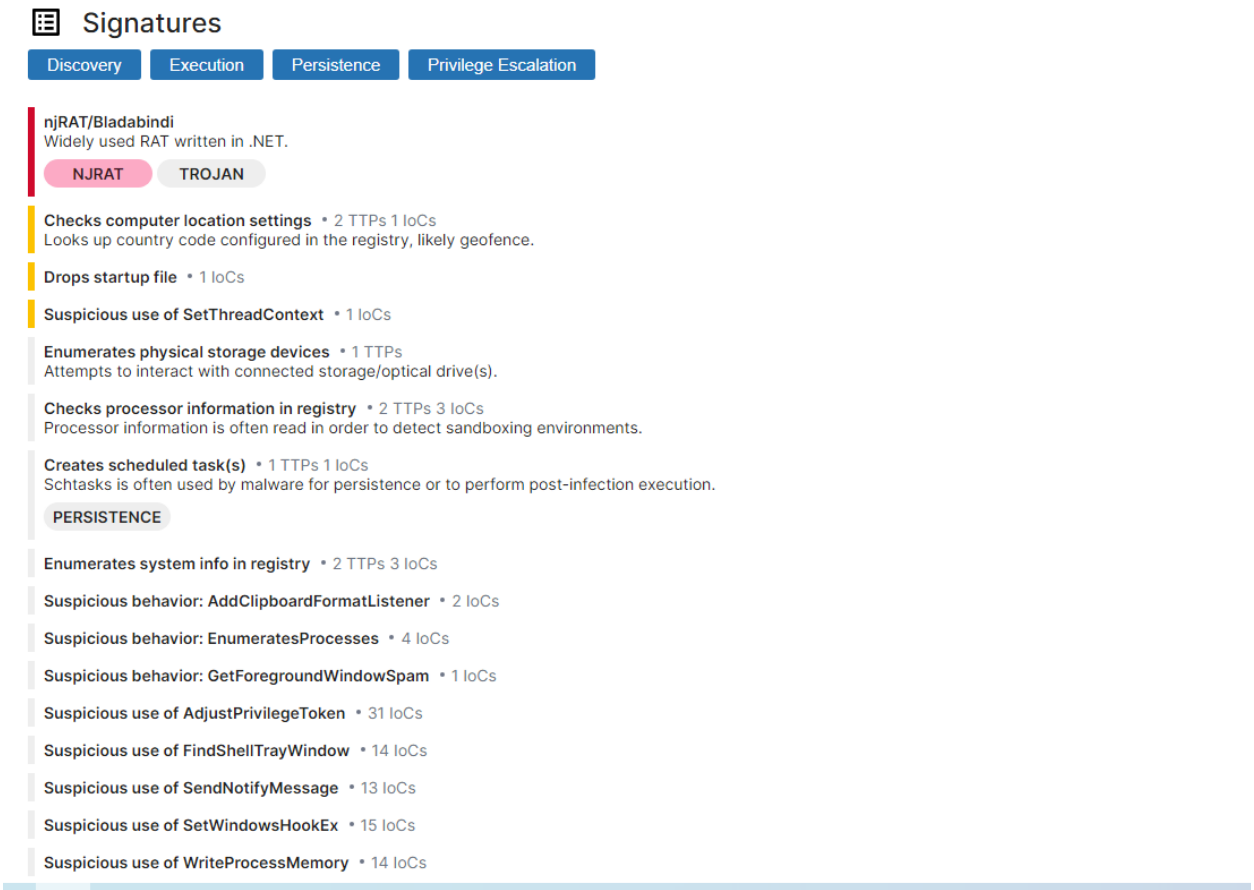


After the execution was complete, this is what it reported:

Clearly a very malicious file.

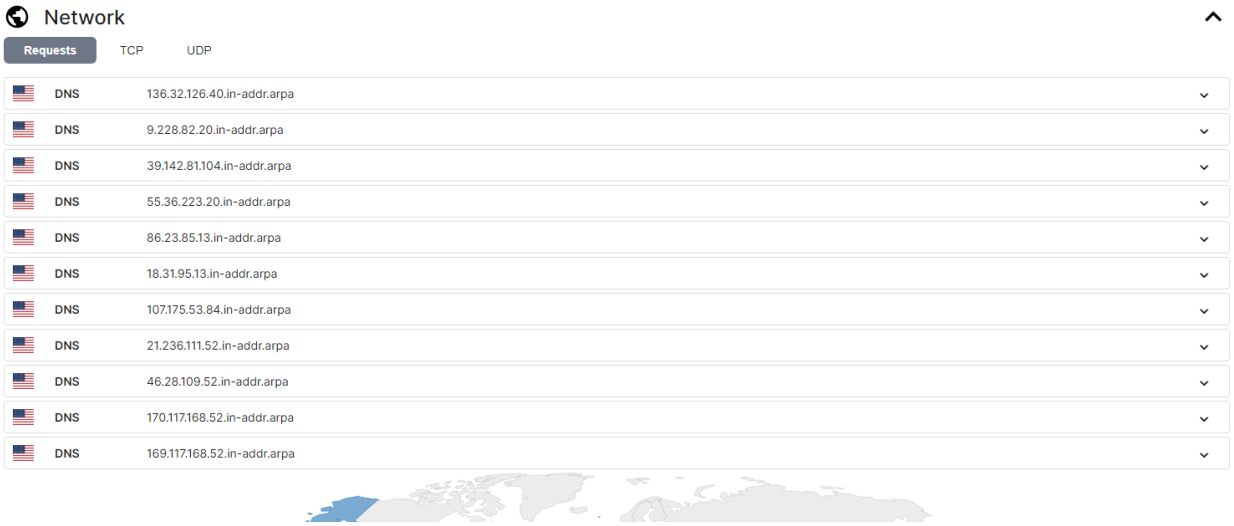


System is compromised and part of a botnet.



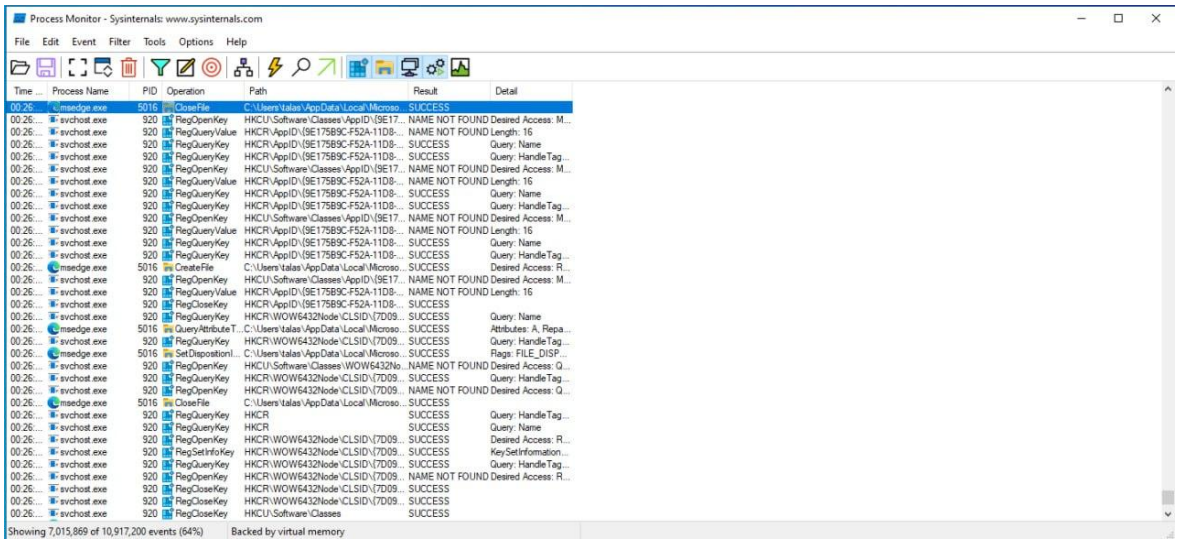
NJRAT: a type of Remote Access Trojan (RAT), a piece of malicious software that falls into the category of remote administration tools, but is used for malicious purposes, often without the knowledge or consent of the affected user. It provides backdoor functionalities and allows for data theft and surveillance.

Also contains TROJAN malware disguised as legitimate software. The file tried to perform many actions that are outside of its role privileges, also tried to suspiciously use and interact with many devices.



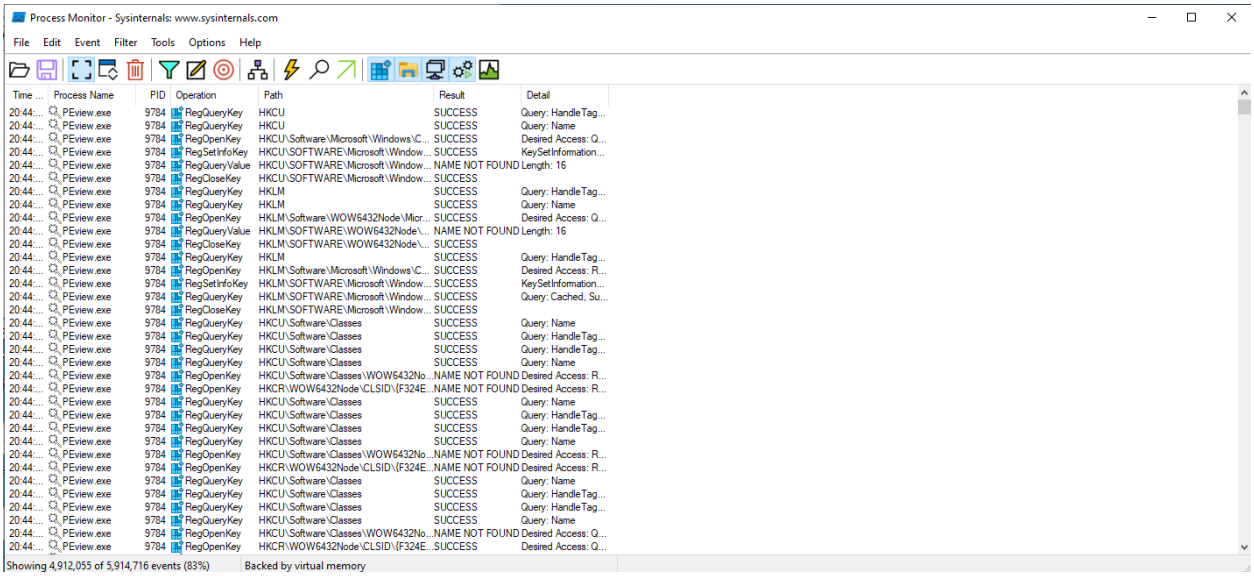
The IP addresses themselves are not suspicious, but the fact that a file is attempting to access a series of IP addresses in quick succession raises many concerns. Additional analysis of network traffic is required to determine the severity of these requests.

- **ProcessMonitor:**

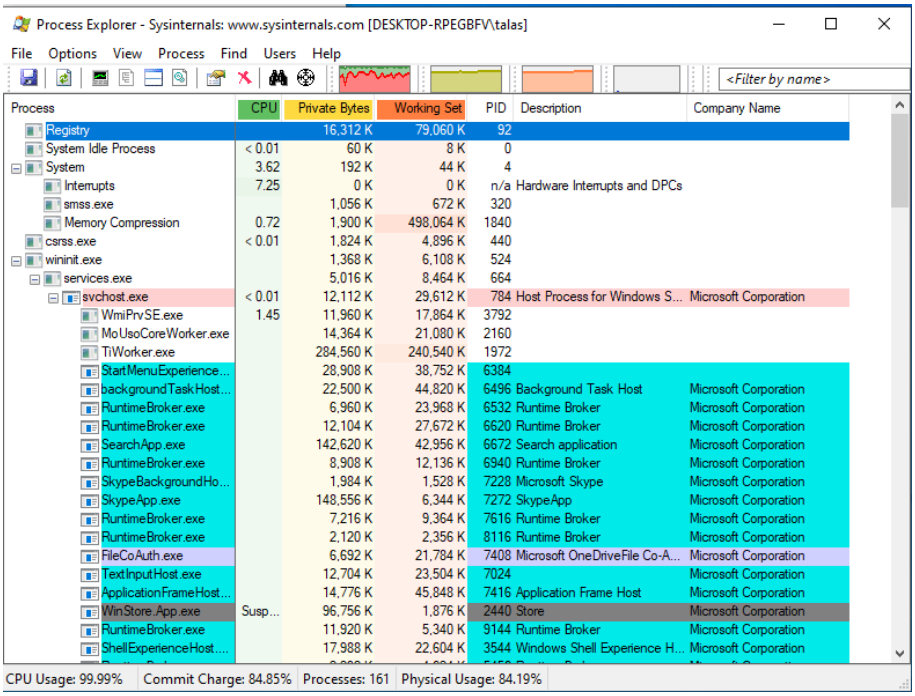


ProcMon captures all events that occur on a file in real-time; for example, if a program is running it captures every single small thing the program does. As we can see from the screenshot, we can even monitor when the file was opened and closed, if that process was successful or not, if it wasn't successful it gives us the reasons it wasn't. Another feature is that it allows you save whatever you want and filter what exactly you want to capture.

For example, when we ran the file on PEview and monitored the behavior on ProcMon, this is what came up:



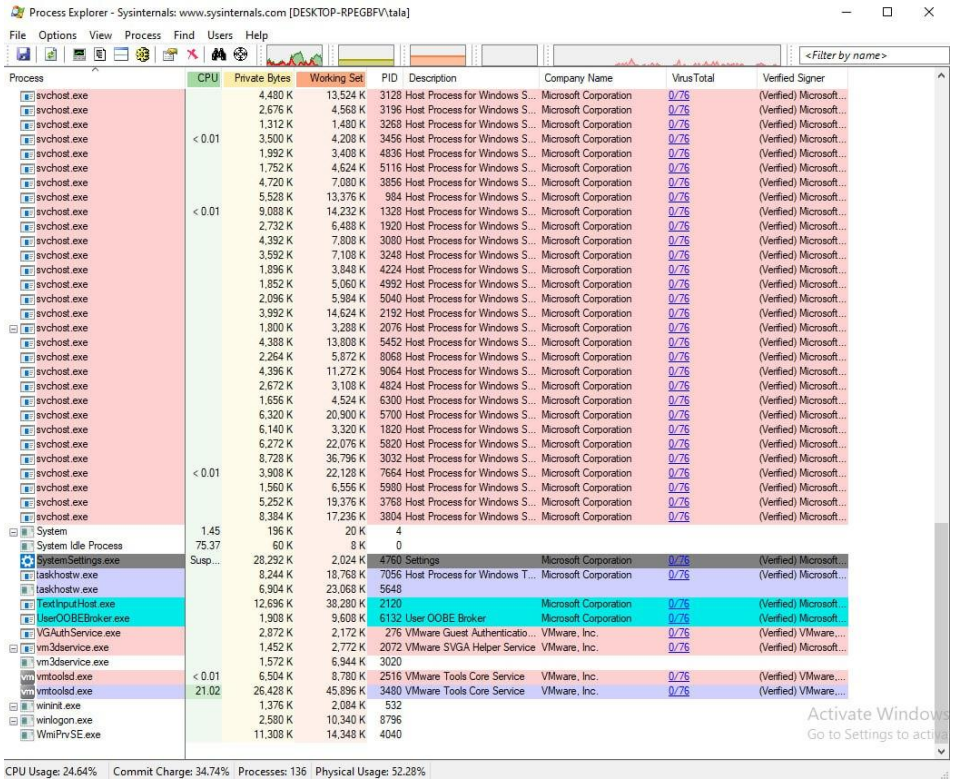
- **Process Explorer:**



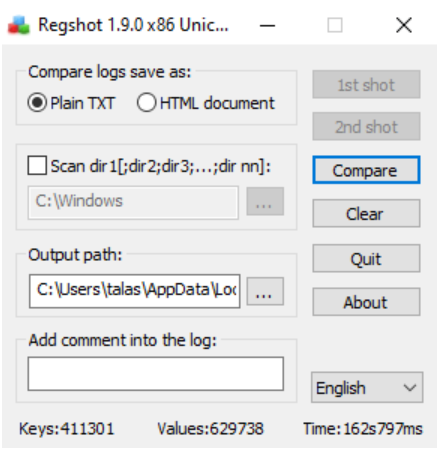
Provides a list of all processes running on the system, helps find out things like the amount of CPU and memory being used. If you are worried about a process, you can easily kill it.

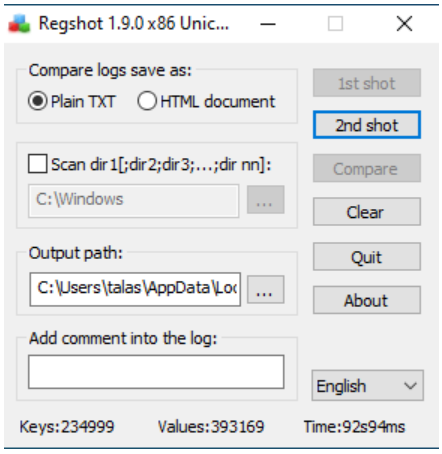
When you click on any process you can see the following things:

- Path
- Command line
- Threads that are associated with it
- TCP/IP (network connections)
- Who owns it
- Strings associated with it



- **RegShot:**





It shows how making a small change will make a big difference. When clicking the first shot it scans the system, then when you click the second shot it scans again and if any change has happened to the system it will detect it and see how many keys were added and how many deleted, and will calculate the total changes. In our case it is 413001 changes in total. The change we made to the system is: check the show/hide [file name extensions] option.

Insights, conclusions, and recommendations:

After using so many tools, here is what we concluded:

- If you are a beginner in malware analysis, tools with a GUI, and online tools are a good way to start. VirusTotal offers a pretty concise report about infected files, it seemed like all the static analysis tools we used just confirmed everything that VirusTotal reported. If you want a general look into the file, we would recommend it. But if you want a closer look into the specifics of this indicative information, you will have to use some PE viewers to really analyze how a file is structured.
- Hash calculators, and even text viewers to an extent are a great way to look into a file, but they are not enough to determine if a file is infected, additional information is almost always required.
- Dynamic analysis will reveal a lot of things about the file static analysis fails to find. The sandbox tool we used really made a big difference and presented a lot of information that would have otherwise remained unknown. Although they typically need more processing power, and the advanced sandboxes are usually paid, we see why they are so essential to malware analysis, as they give critical insight into files that no other tools can provide.

***References:**

- *PE Format*: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format#machine-types

- *File Entropy:* https://practicalsecurityanalytics.com/file-entropy/