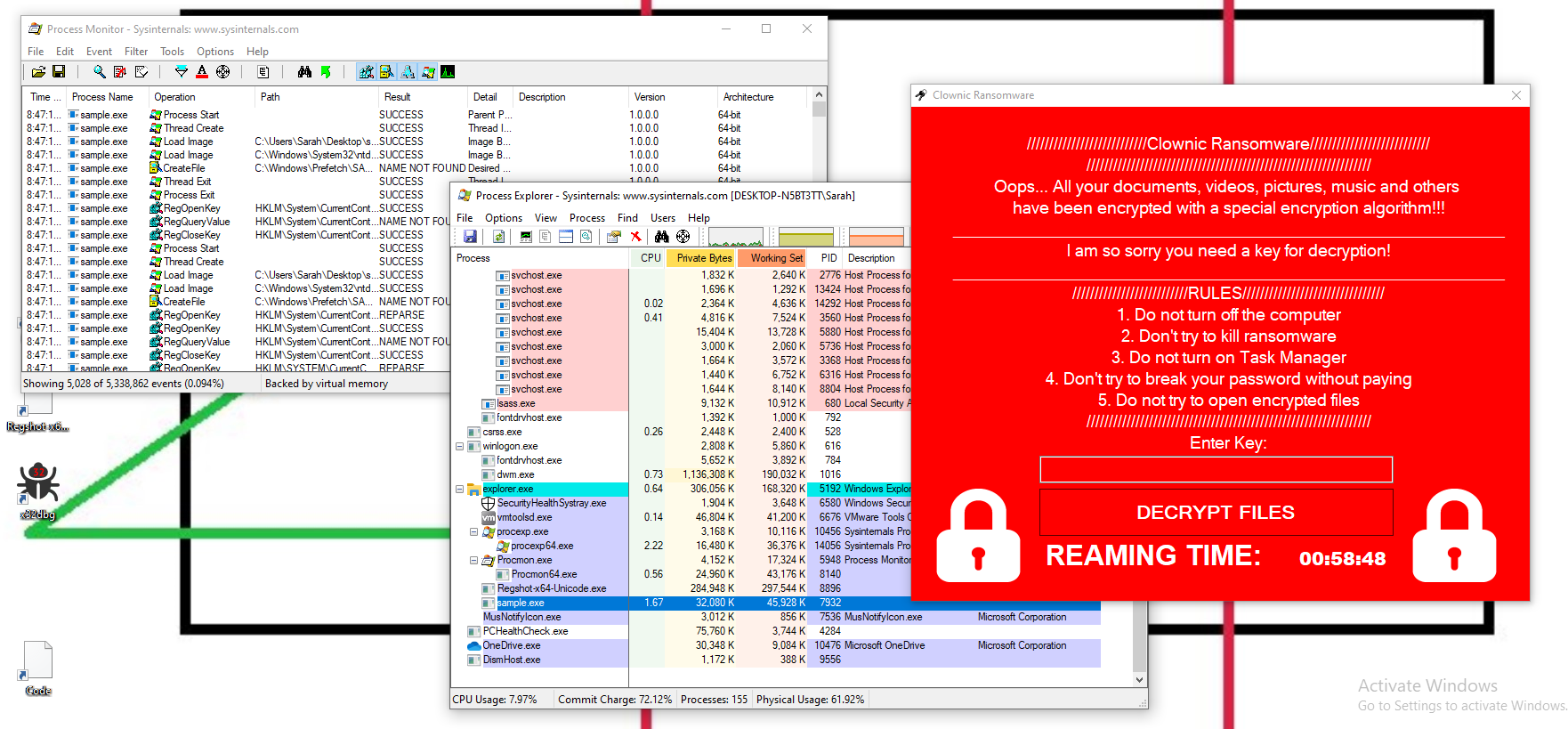
Sarah Dill

CSEC 202 — Reverse Engineering Fundamentals

Section 1

14.12.21.

Analyzing Clownic Ransomware



# 

# 

# Introduction

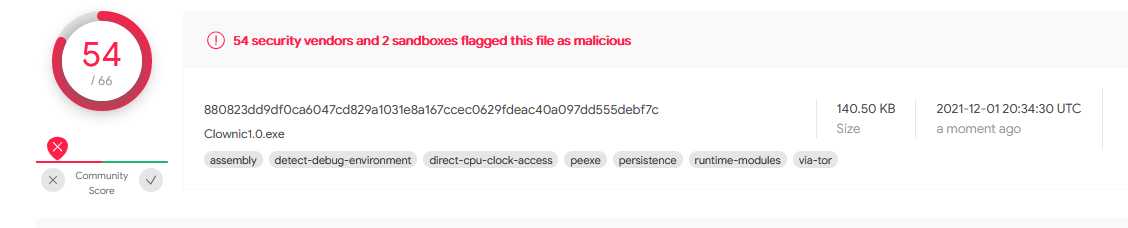
Clownic Ransomware is a relatively recent malware specimen, with the earliest references being from August 2021. Clownic is a piece of ransomware that encrypts all files and demands payment to receive a key for decryption. As this specimen is still fresh, not much is known about it, so it is time for me to do some digging and analysis of my own.

It is interesting to note that as soon as I unzipped the file, Windows Defender identified the ransomware and shortly after deleted it. So in order to be able to analyze the malware, I needed to turn off real-time protection.

# Basic Static Analysis

## VirusTotal

Before getting started doing my own analysis, I found the VirusTotal statistics. And, as is shown very clearly, this file is malicious.



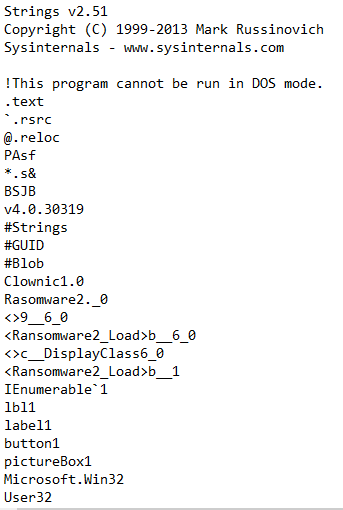
Some of the tags associated with this malware give some hints as to possible behavior of the malware, such as “detect debug environment”, “direct cpu clock access”, and “persistence”. These tags alone suggest a very aggressive malware specimen that may provide some trouble for me later.

## Strings

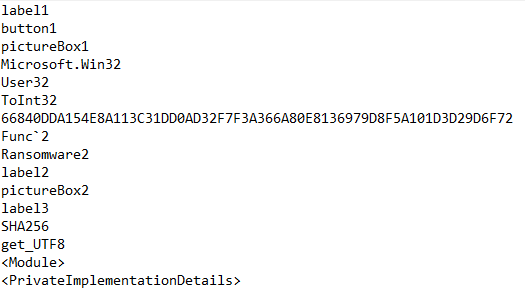
The first step in dissecting this malware sample is to run it through strings. In order to improve readability and make things simpler, I renamed the malware sample to “sample” (as before it was named as a long sequence of seemingly random characters that had no meaning), and I sent the output into a text file called sample.txt, as shown below.



When I opened the file, there were some interesting strings presented right off the bat, including strings containing “Ransomware” and a few GUI-related strings, such as button1 and label1.



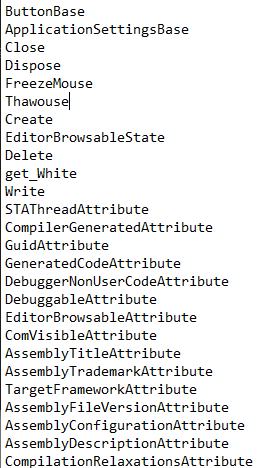
Shortly after, a long string of numbers and letters showed up, which could be indicative of some sort of encryption algorithm at play.



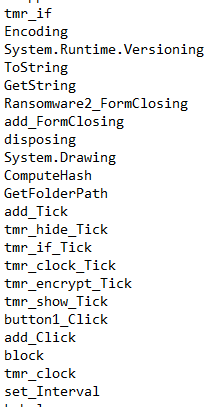
This hypothesis is further supported by strings that appear not too shortly after, including SHA256, bytesToBeDecrypted, bytesToBeEncrypted, DecryptFile, and EncryptFile.

|  |  |  |
| --- | --- | --- |

Other strings indicate that there may be some debugger detection going on, such as the DebuggerNonUserCodeAttribute and DebuggableAttribute, as shown below.

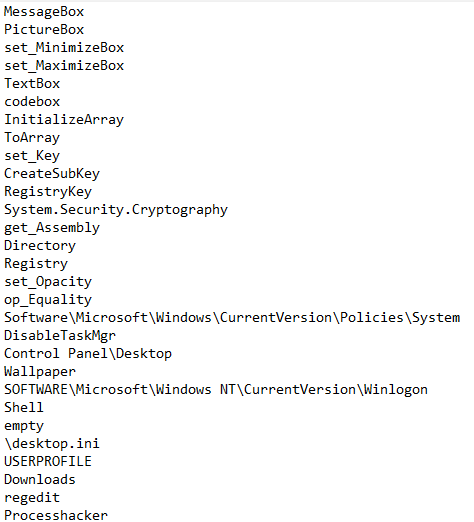


There are also many strings present including “tmr”, which is indicative of some sort of timer or behavior involving the CPU clock (which makes sense from the VirusTotal scan).

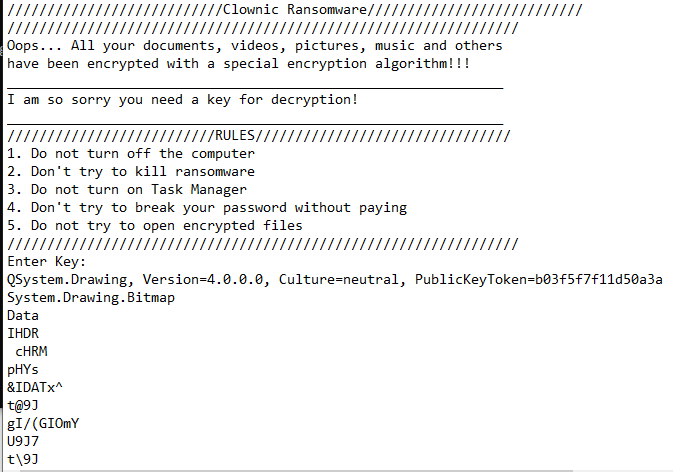


The “ToString” and “GetString” functions are also interesting. Perhaps this has something to do with file names or encryption/decryption.

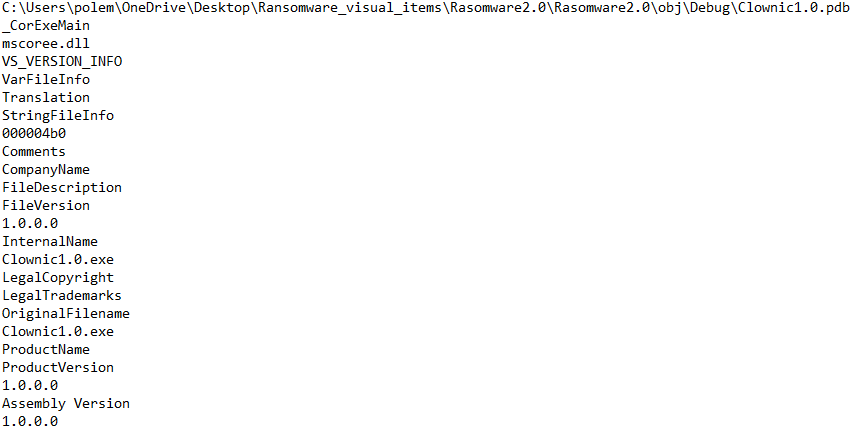
There are also several strings that indicate modifying registry keys, the wallpaper, and disabling the task manager. The “\desktop.ini” string is also interesting. Why would the malware need this string? The purpose of this is unclear.



The actual ransomware text is also present:



There’s also a set of strings that include debugger information, including the full file path to the .pdb file, a dll filename, a possible function call (\_CorExeMain), and some general information about the file itself.

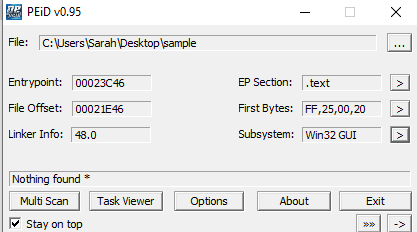


Overall, if you didn’t know that this was a piece of ransomware before, you definitely do now. Most of the strings point back to typical ransomware activity. It was strange that there weren't a whole lot of “garbage strings” found, which could indicate that the ability to see these strings was very intentional. Because of the sheer number of strings found, it seems that this file isn’t packed much if at all, but the fact that the number of “garbage strings” wasn’t very high indicates that something suspicious may be going on.

## 

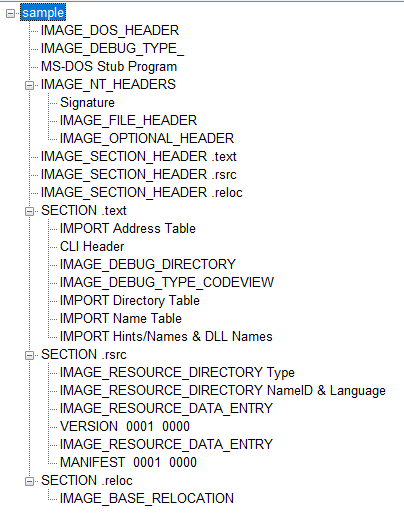
## PEiD

In order to confirm my hypothesis that this file is not packed, I opened it in PEiD. Nothing of interest showed up here.



## PEview

To further support my hypothesis and to get a glimpse of the possible function calls that may be going on, I opened the file in PEview.



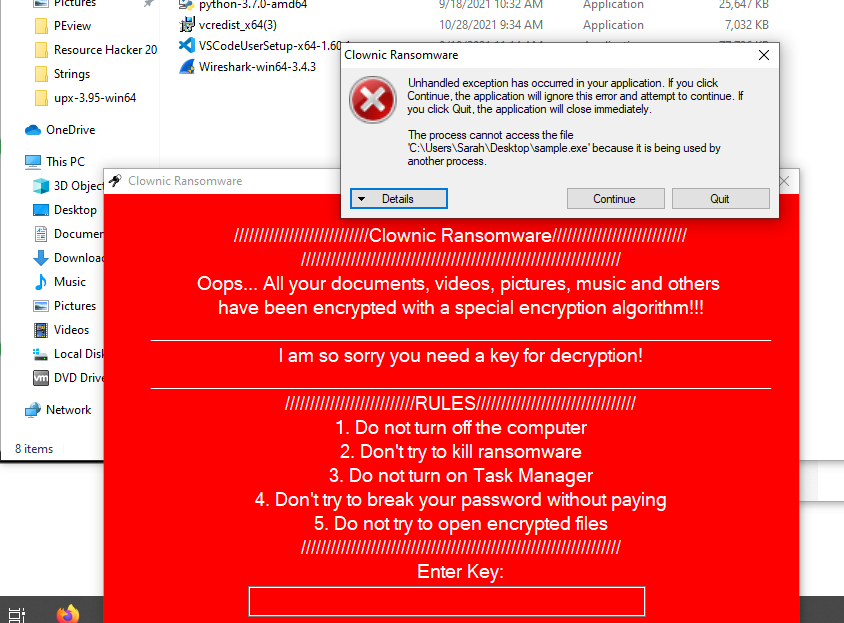
What’s interesting here is the “CLI Header” under the .text section, the reference to various debug sections, and the lack of .rdata and .data sections. What’s also interesting is what’s listed in the “IMPORT Address Table” under the .text section (which is strange in and of itself, since the table is usually in one of the data sections).



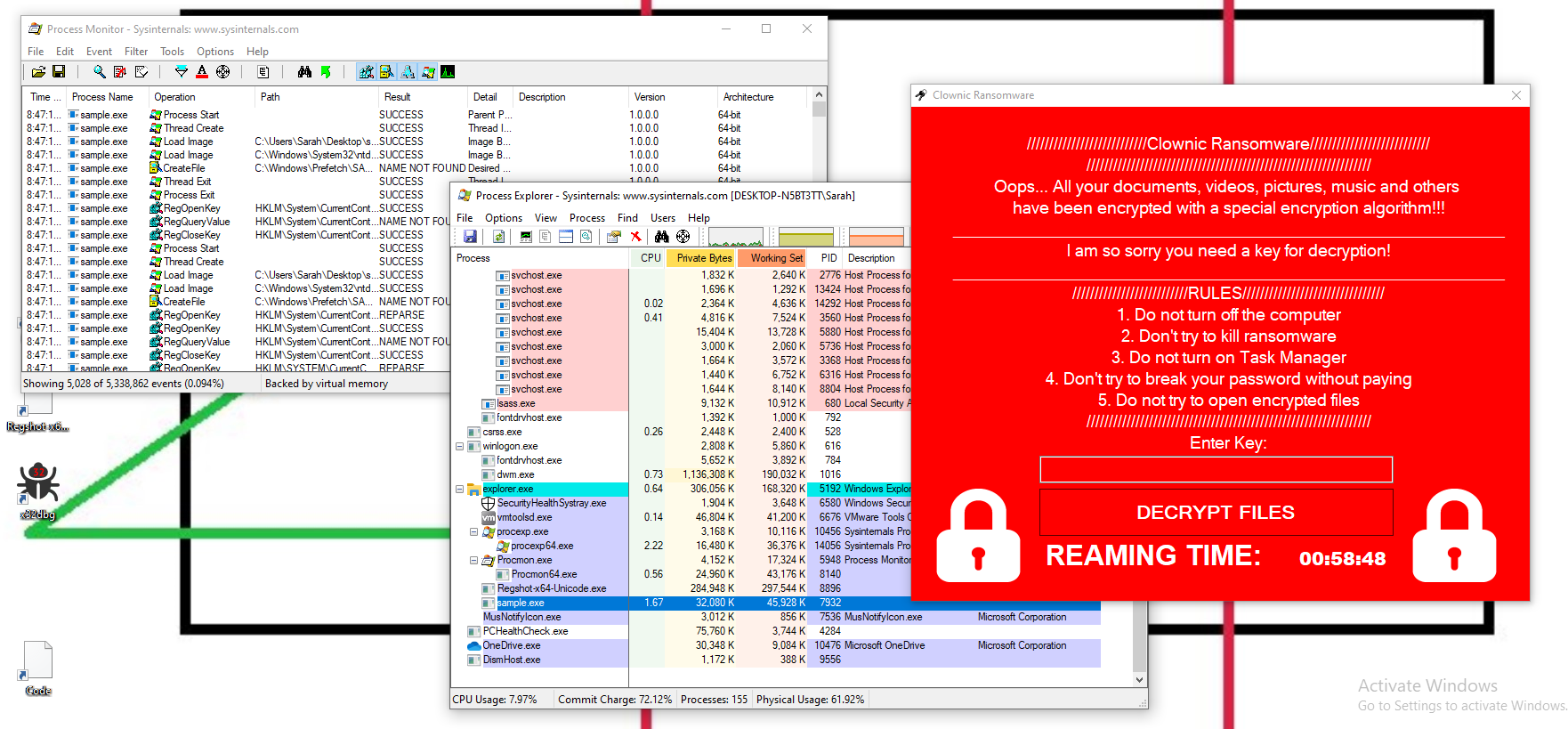
There’s hardly anything in here! Not even GetProcAddress and LoadLibrary, which are two functions that cannot be hidden in obfuscation/packing. There is some possible strong obfuscation and/or packing. There isn’t even a binary or dll to extract from here to indicate a hidden binary that executes the actual code. Something very fishy is going on.

# Basic Dynamic Analysis

Since the file type wasn’t specified, I had to change the file type to .exe, and then the malware was able to execute. When run, the computer seemed to freeze for a few seconds before a dialogue box popped up:



This was shortly followed by the ransomware application itself appearing.

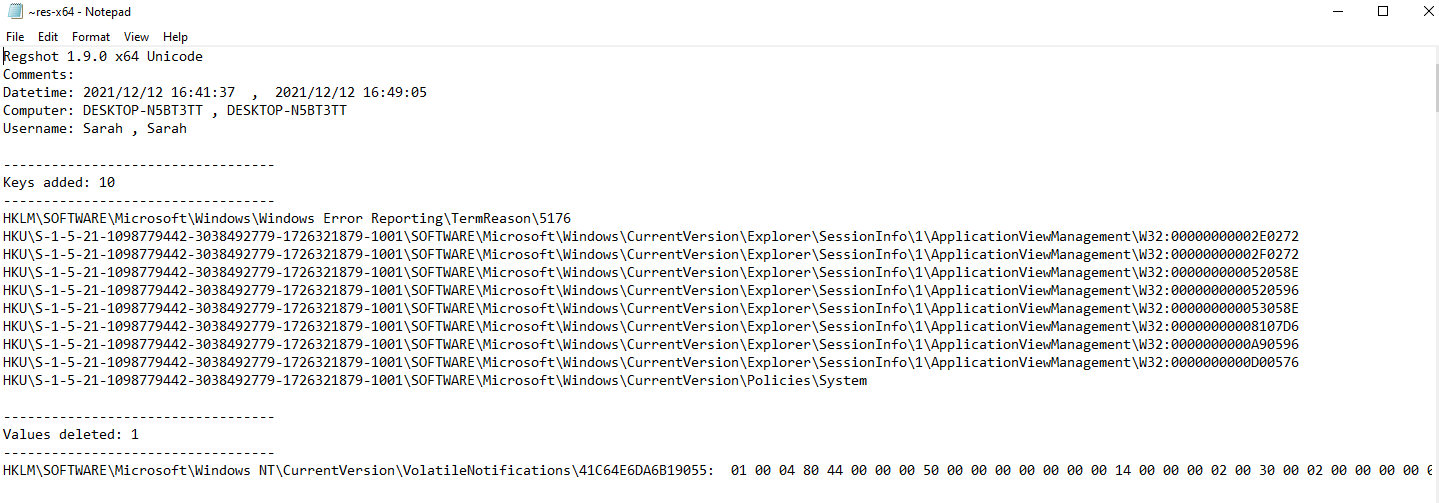


An interesting point to note is that just with this GUI, while the rules state not to try to break the password without paying, there is no clear way for a victim to even try to pay, which implies that the malware author may have had no intention for the victim to get their files restored. The poor English text also indicates either sloppy programming (which is doubtful considering what I saw—or rather, *didn’t* see—in PEview), or a non-native English speaker author.

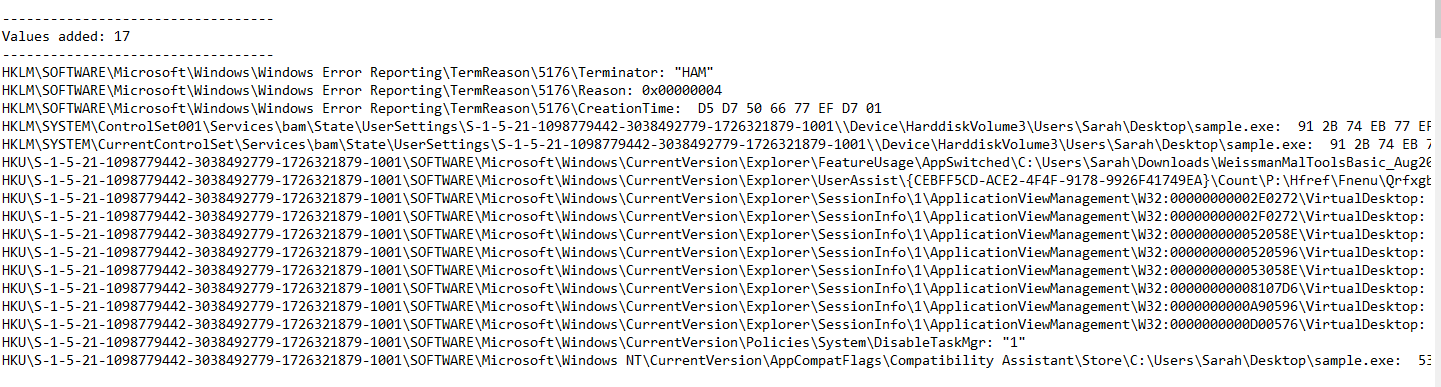
## Regshot

Running Regshot produced some interesting results. In total, there were 10 keys added, 1 value deleted, 17 values added, 50 values modified, 15 files added, 11 files deleted, 42 files modified, 1 folder added, and 1 folder deleted, yielding 148 total changes.

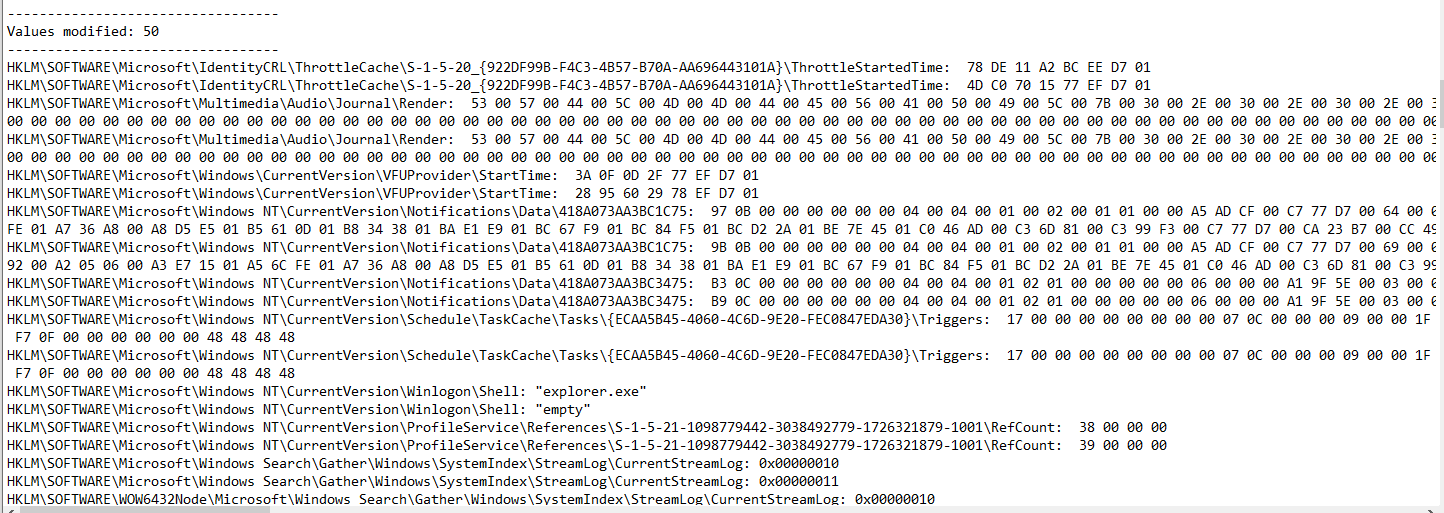
In the “Keys added” section, most keys involved keys in the “Application View Management” section under “Explorer”, which could be related to the malware that just spawned.



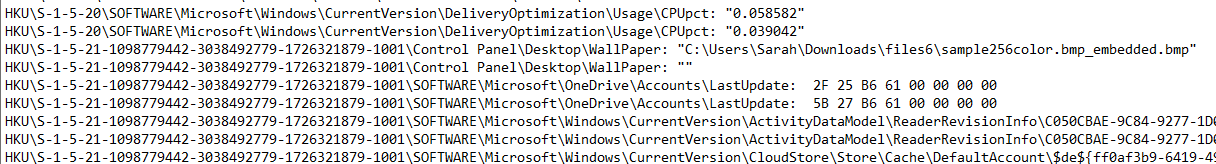
The most interesting line in the “Values added” section was the one ending in “DisableTaskMgr: “1”, which most likely disables the task manager. After a simple test of trying to open the task manager, I got an error message indicating that the task manager had been disabled by my administrator. The first several characters match the last key listed in the “Keys added” section, which was most likely a set-up for adding the ability to disable the task manager.



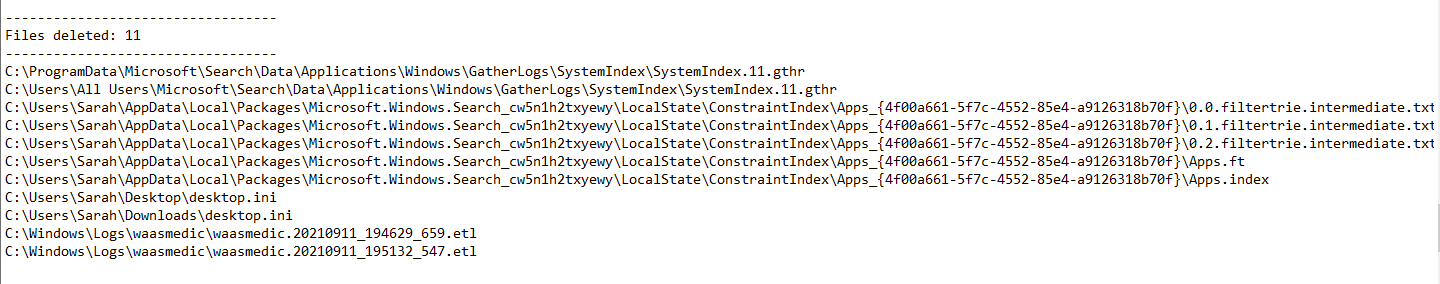
In the “Values modified” section, there were many values that had some hex listed after each value, which is where the modification occurred, as the strings containing file paths or key names had no visual change.



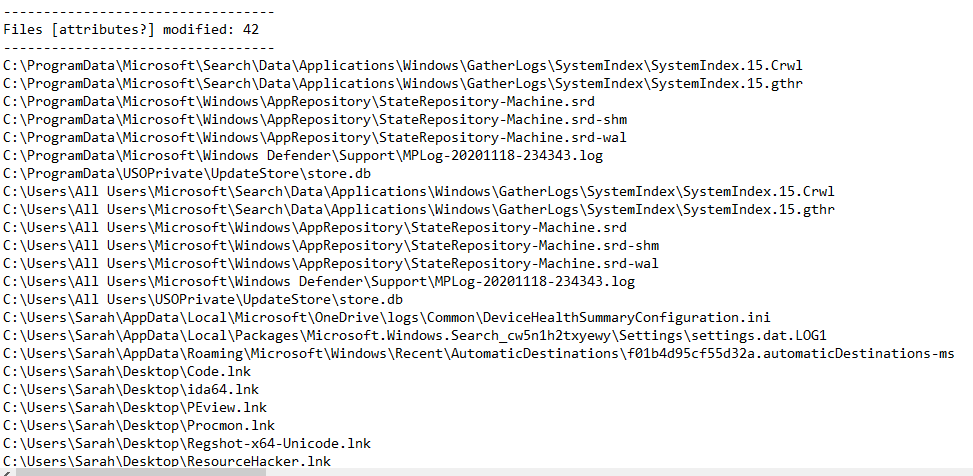
One interesting modification to note was the wallpaper file path, which was changed to an empty string. Despite this change, there was no visual difference in the wallpaper. Whether this was intentional or a feature of some possible VM detection is unclear.



In the “Files deleted” section, we can see that “desktop.ini” is deleted both from the desktop and downloads folders, which has the effect of changing folder settings back to default. Why the malware did this is unclear, but the behavior is interesting to note.



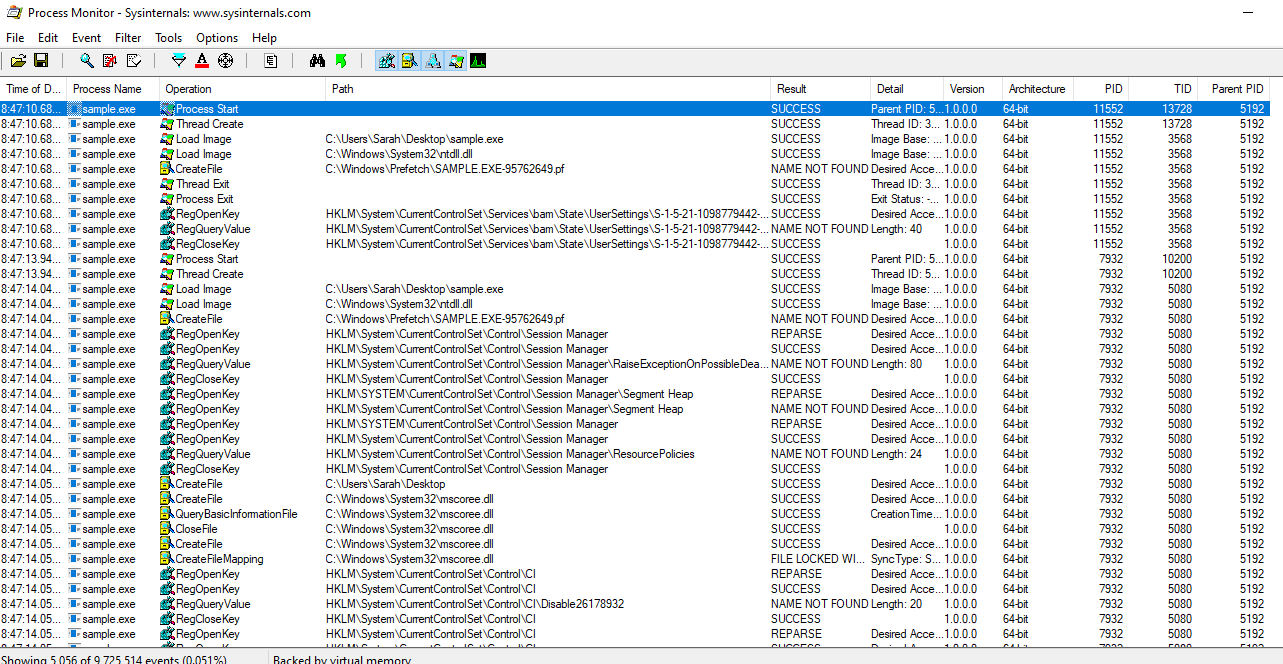
It is not at all surprising that there are many files modified, including the shortcuts I had on my desktop, which aligns with the hypothesis of encryption going on.



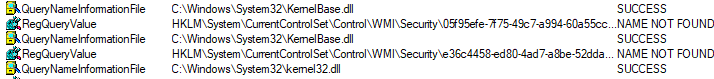
Overall, most behavior captured by Regshot matched the strings I saw in my earlier analysis.

## Process Monitor

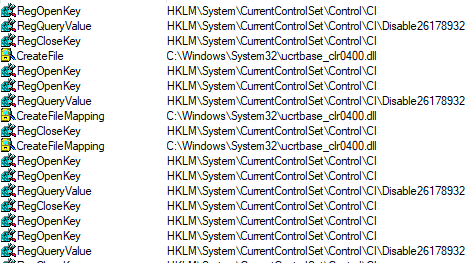
Using the executable name as the filter, I was easily able to see at least most of what the malware was doing. At a first glance, there were over 5,000 events created by this one executable, with operations ranging from manipulating files to querying registry keys, which also lines up with what was seen in the Regshot text file.



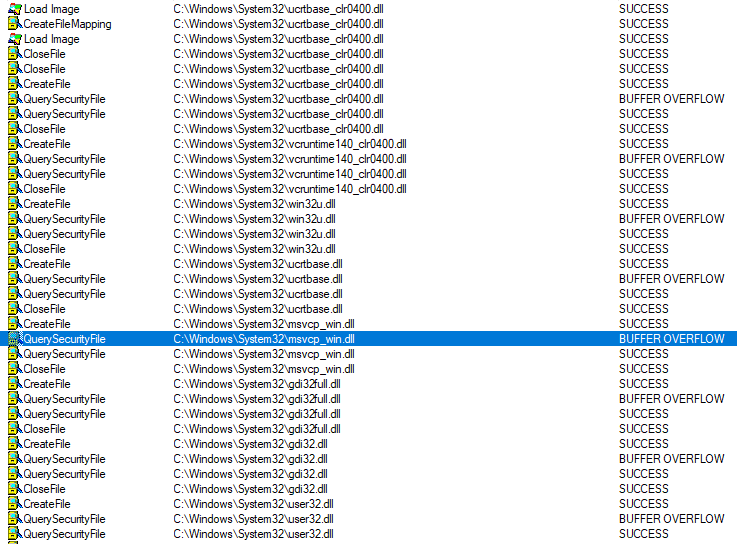
There were some events that queried information about KernalBase.dll and kernel32.dll. It is unclear what this information was used for.



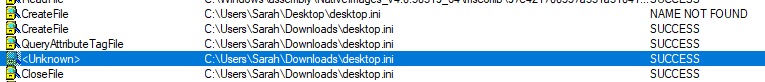
As mentioned, many events dealt with registry keys. It seems to be a trend that each key is opened, a value is queried, and then the key is closed.



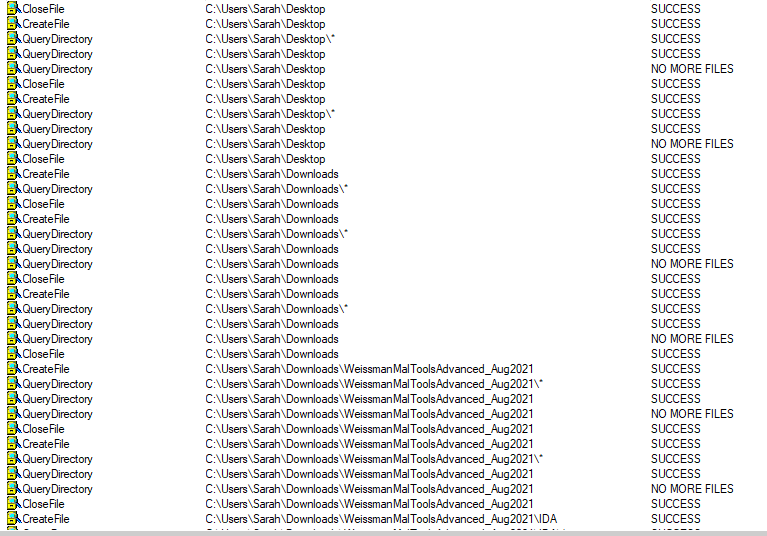
We can also see that many dll files are being created and accessed, each seeming to follow some sort of pattern of being created, queried, then closed. In the “result” column, we can see that the first time the dll is queried that it resulted in a buffer overflow.



There is also some activity regarding the desktop.ini file. It is interesting to see that there is one operation that is listed as “<Unknown>” that was successful before being closed.



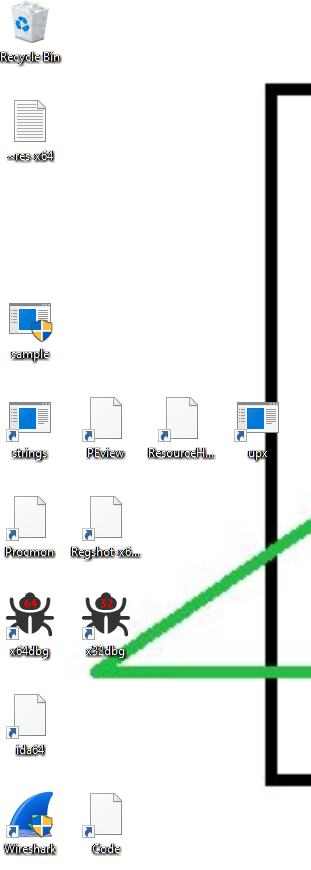
This appears to be where some possible encryption may be happening, as the malware is going through each file and continues until there are no more files.



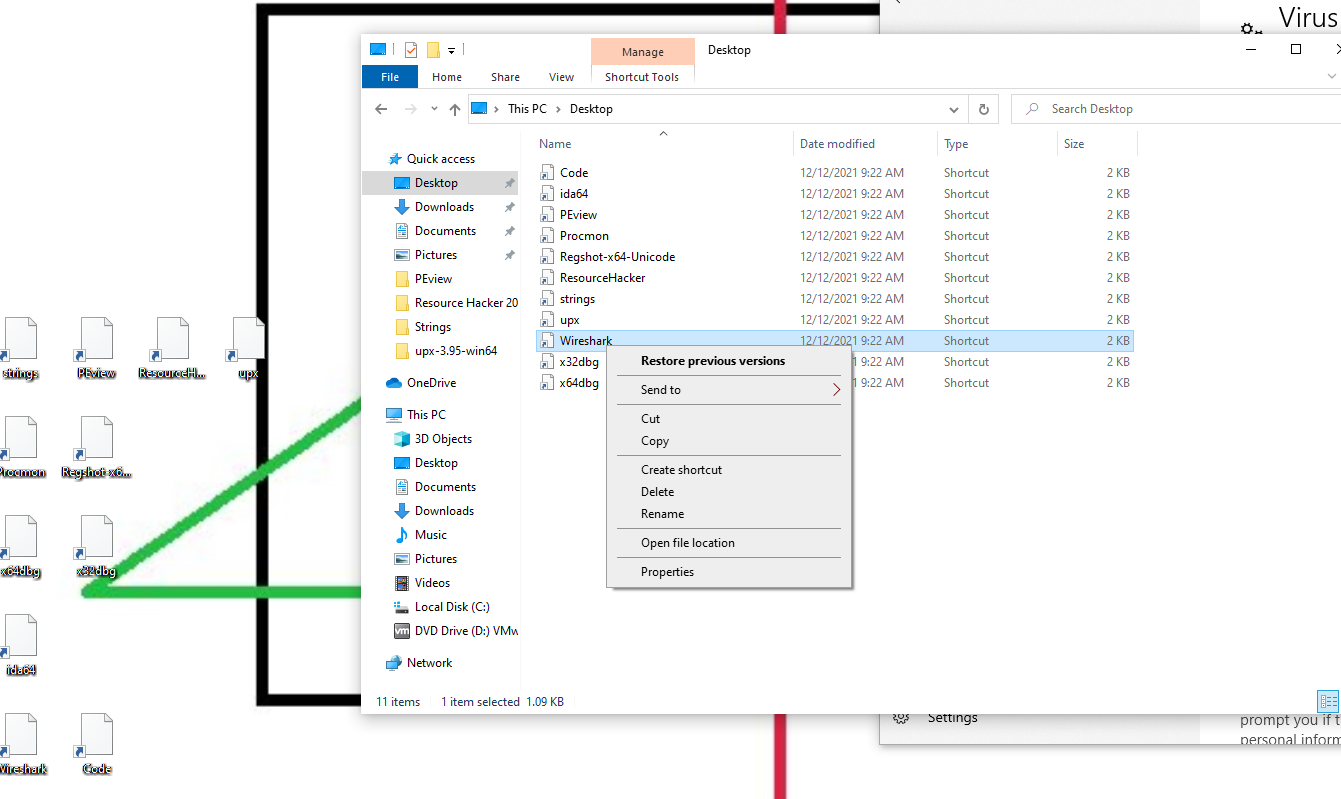
# 

## Testing Encryption

My hypothesis that this malware encrypts files has yet to be proven, so let’s see what damage has been done to my files.



In this screenshot of my desktop, we can see that some icons have the default file icon attached, and some have their own unique icon. This could be a result of the desktop.ini file being removed. When I clicked on an icon that had its associated image removed, nothing seemed to happen. When I right-clicked the file, there was no option to run the executable.



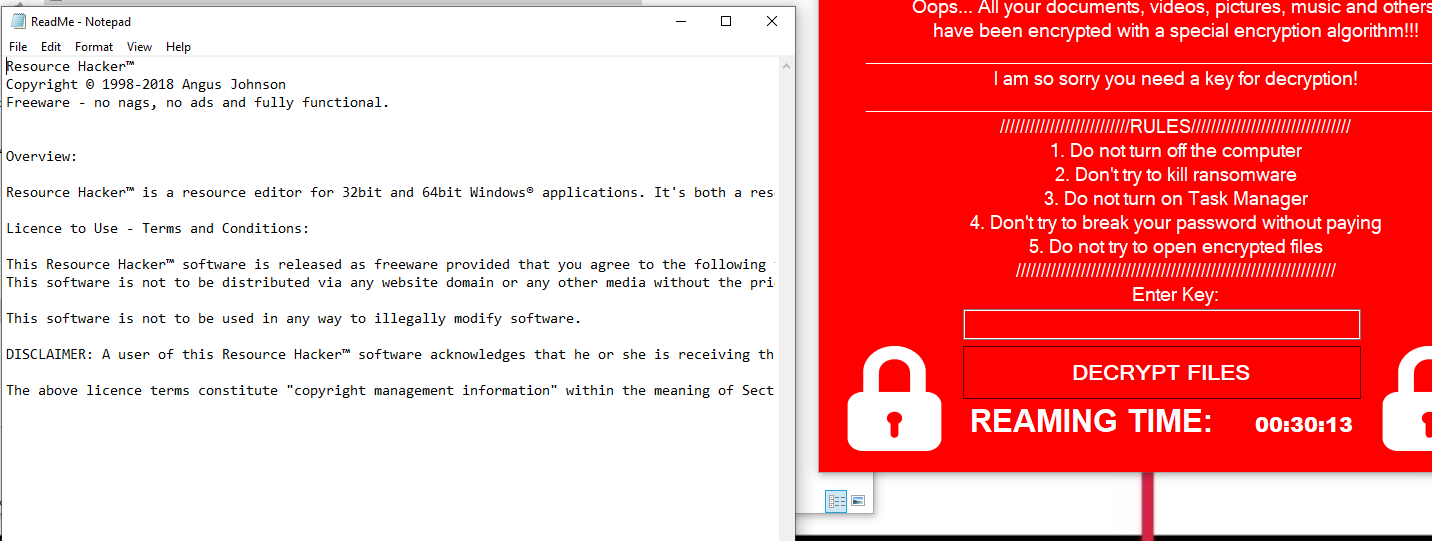
When I tried restoring a previous version, Windows did not see any previous versions.

Clicking on an icon that appeared to be untouched actually ran the program, as shown below.

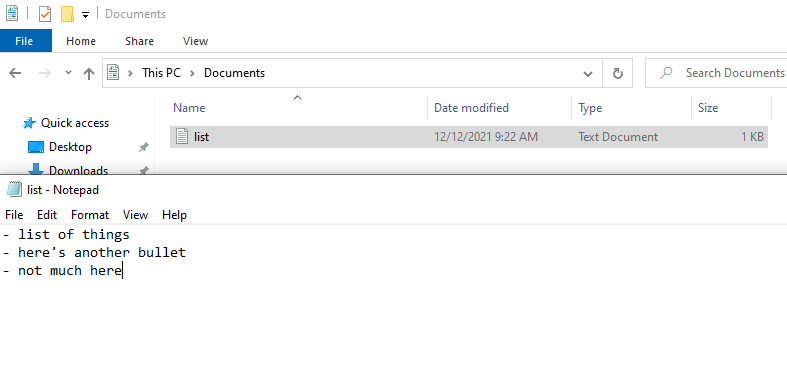
# 

We can also make the assumption that the malware does not encrypt newly-created files, as the text file from my Regshot analysis was not encrypted.

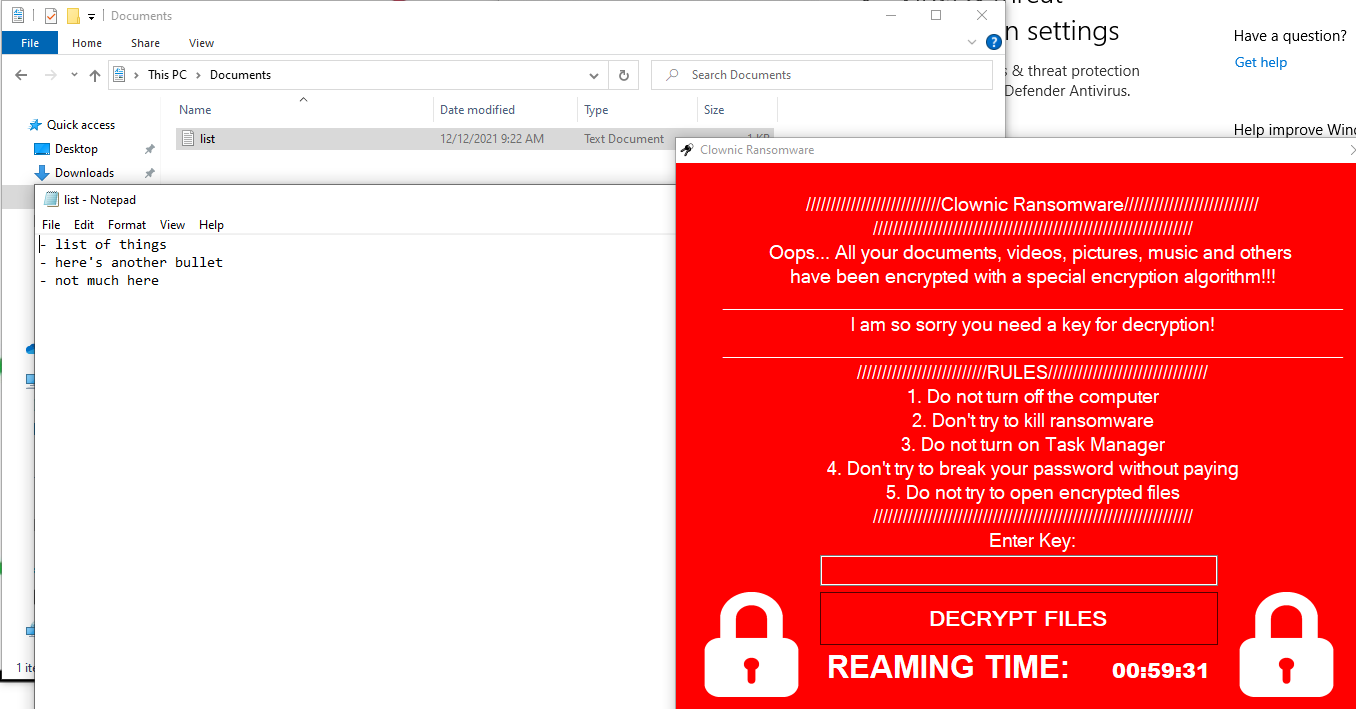
The claim that everything has been encrypted was a false one, it seems. Clicking around more, it appeared that nothing else was harmed besides those desktop icons. As you can see, this text file that was present before the malware execution shows up fine even after initial execution.



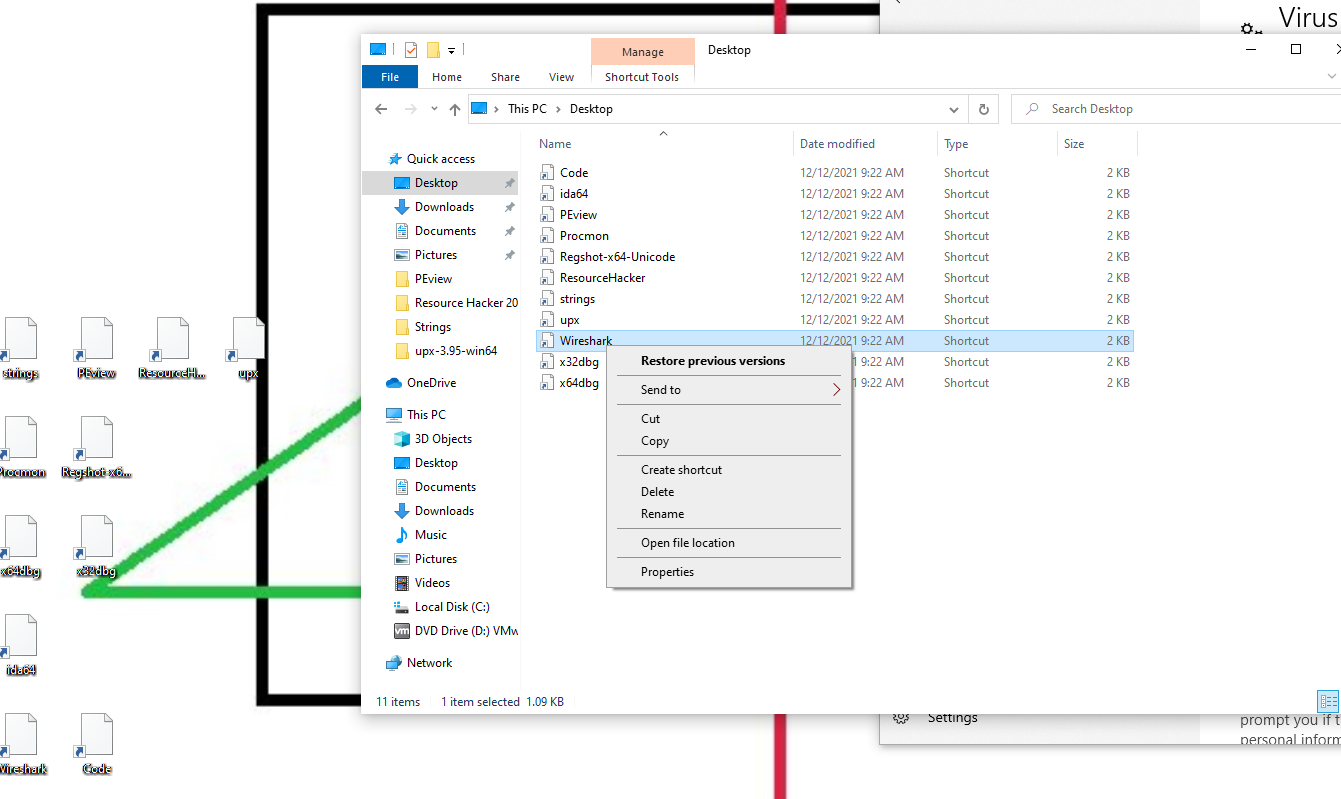
To confirm this, I decided to do some testing. There were no text files previously in the Documents folder, so I created a dummy text file with some random text inside.



When I ran the malware this time and attempted to open the text file, it opened just fine, and the contents were not messed with.

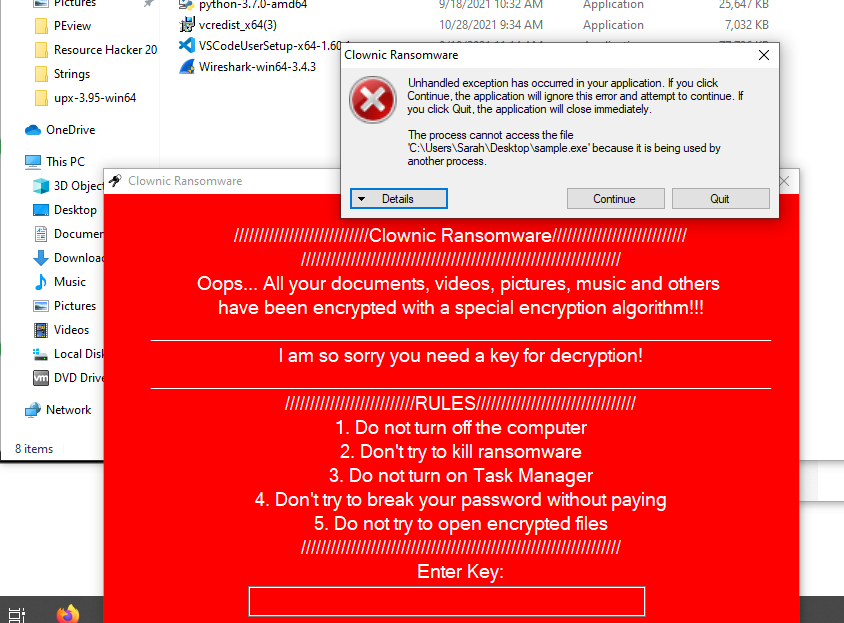


However, on this second runthrough of the malware on a clean VM state, running the malware from the Downloads folder yielded different results regarding the files on the desktop. Instead of only partially encrypting the files, it seemed to encrypt all of them.



This partial encryption could be indicative of VM detection, so the malware doesn’t do everything it was built to do in order to prevent reversing or to seem less damaging than it actually is.

In another runthrough, I decided to mess with the original pop-up dialogue.



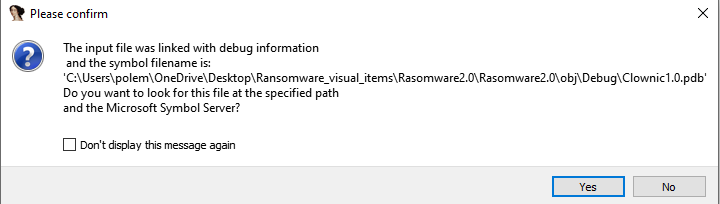
In all the other times I ran the malware, I clicked “Continue”, which kept the malware up and running. When I clicked “Quit”, the malware shut down; however, in doing so, my desktop icons were still left encrypted.

# Advanced Static Analysis

## IDA

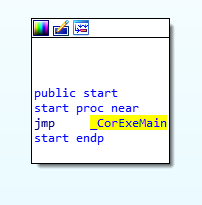
IDA originally did not know what the file was and gave me 3 different options. The one that yielded the most interesting results was the option to analyze it as a PE file.

The first thing to appear was this pop-up, asking about provided debug information, where a familiar string appeared.

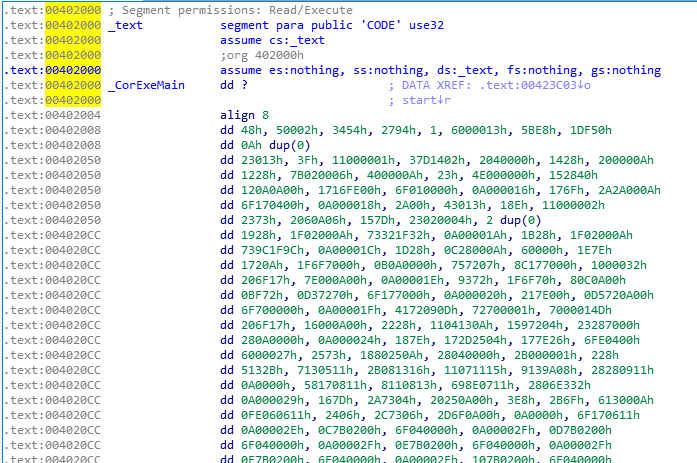


The hypothesis that this was some debug file path was correct; however, since we do not have the actual file, there isn’t much to analyze.

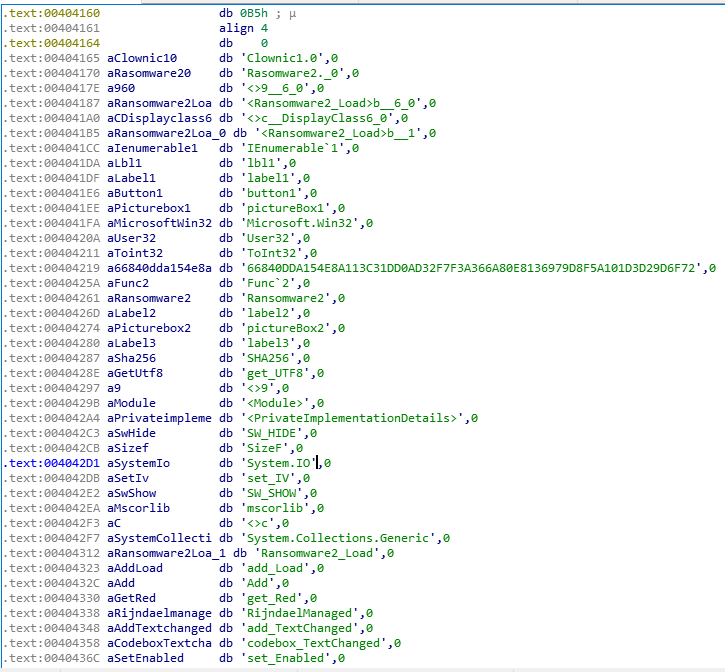
First, we get a very small function with a single instruction: jmp \_CorExeMain.



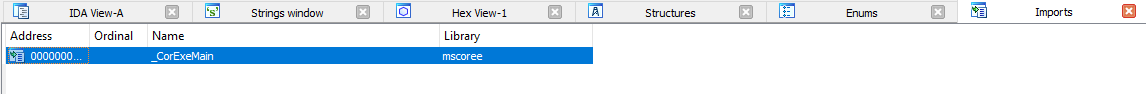
When we navigate to see what’s there, all it points to is a bunch of text that doesn’t appear to be useful.



Scrolling through these many lines of text, we can see that there is a list of function and variable names, some of which did not appear in our original Strings analysis.



When looking at the Imports section, we see that there is one import from the mscoree library.



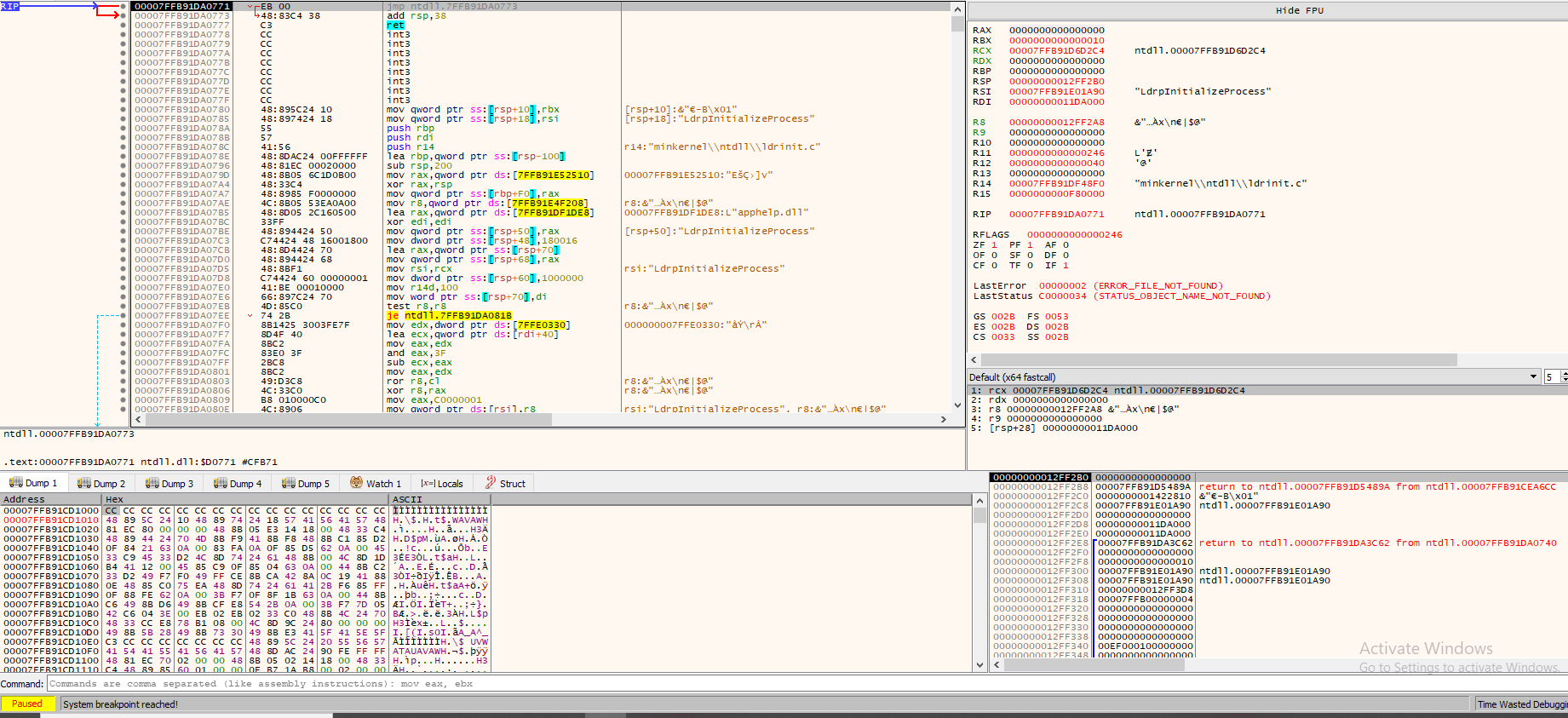
Other than these vague clues, there isn’t much else to analyze here, especially without the debug file. It seems that this malware author did not want anyone reversing this executable, and we are left with a shell of what the malware is. Now, let’s see what we can find in our advanced dynamic analysis.

# Advanced Dynamic Analysis

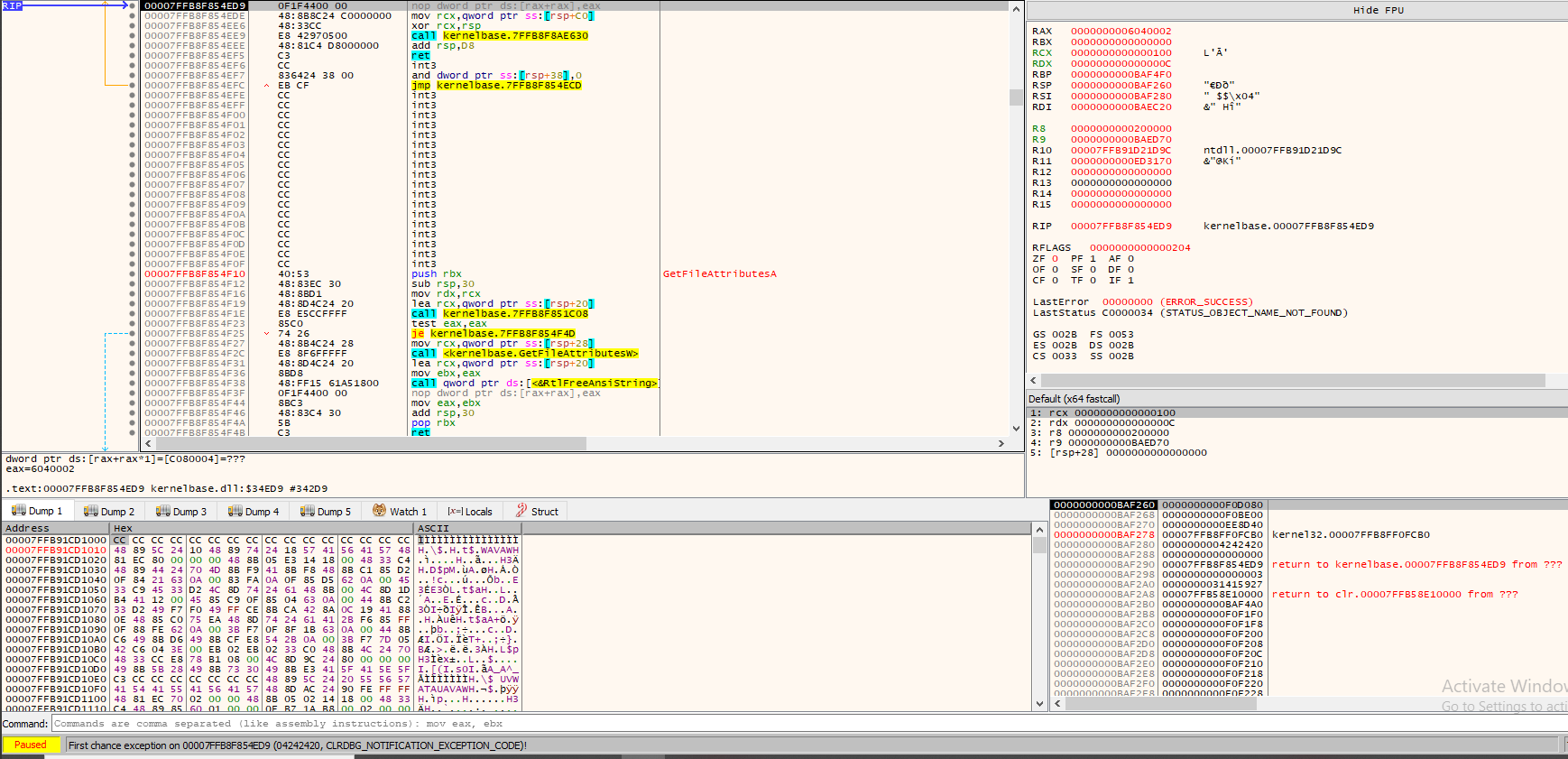
## x64dbg

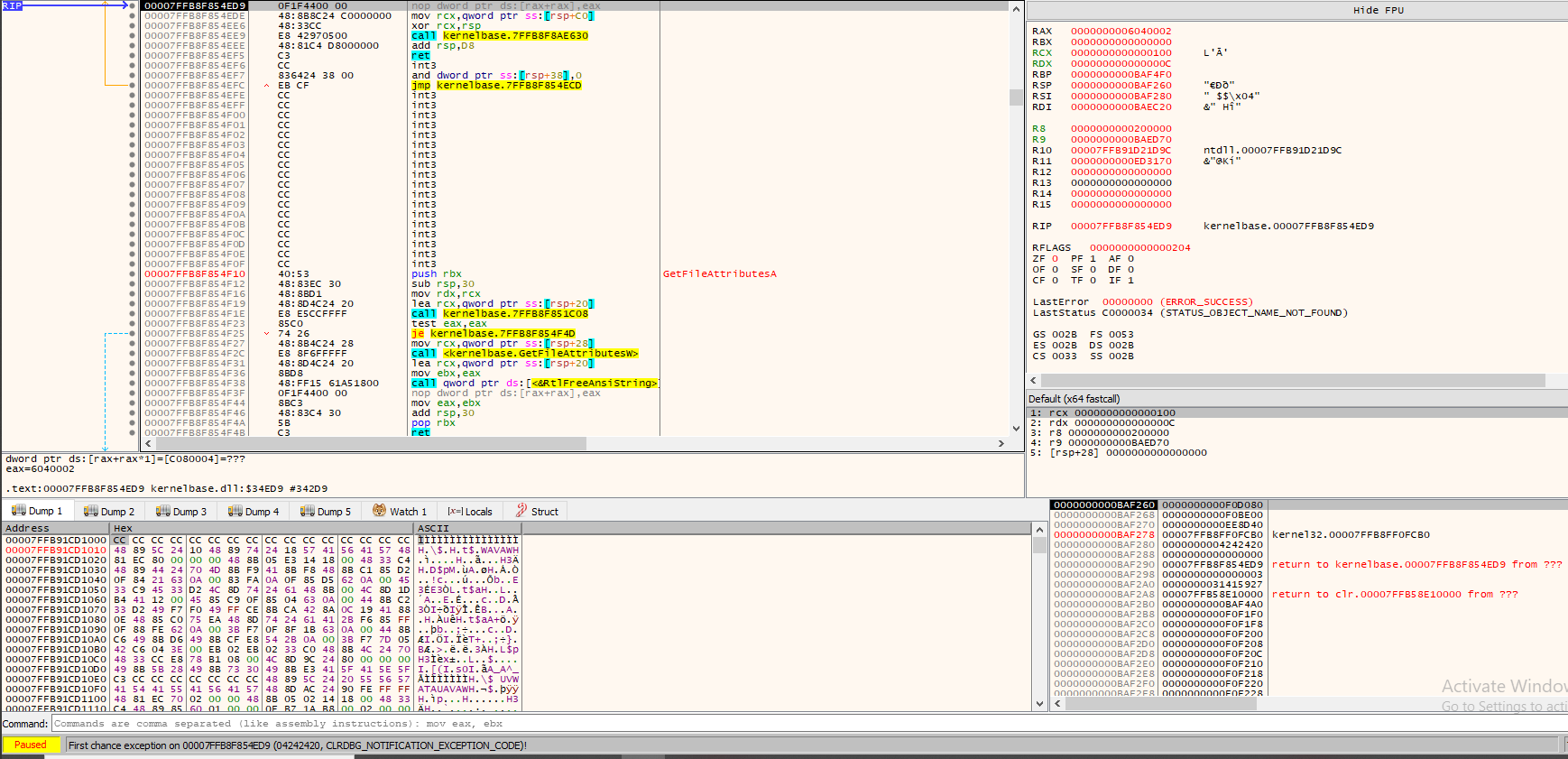
We know that this executable is 64-bit from Process Monitor, so I opened the file in x64dbg. The longer addresses and the use of the larger CPU registers confirm this.

Initially brought up is what seems to only be a few lines ending in the ret instruction.



Let’s move out of the initial set-up and go straight to the main function of the executable. When first running the program, it leads us here with an exception thrown:

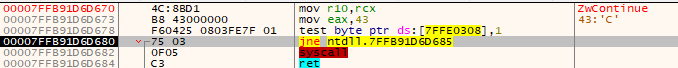




When I continued running the program, everything seemed to freeze, and my mouse clicks did nothing. This is most likely the malware’s doing, likely serving to prevent the user from doing anything while the malware encrypts all the files, with the additional purpose of preventing others from debugging and reversing the malware.

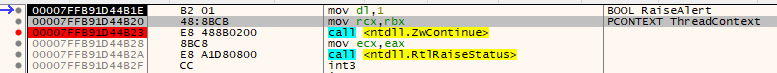
After resetting the VM, it was time to see where exactly this exception is thrown and see perhaps where we can bypass this.

After lots of clicking around and some digging, the ZwContinue call caused the exception to be thrown.



This function runs a few times before going to where the first F9 hit went and throwing the exception.

Before this call, we can see that two values are being moved into registers: the value “1” into the dl register and the value stored in rbx is being moved into rcx. Since rcx and rdx are the first two registers that are used for function arguments, it is likely that these values are used in the ZWContinue function. Looking up this function, I found that these are indeed parameters for the function: PCONTEXT ThreadContext and BOOL RaiseAlert.



It isn’t clear exactly what’s going on here, but this is definitely some sort of anti-debugging technique—all before even reaching the main function. In order to get around this, other tools are needed, which brings my analysis to a halt.

# Summary

Clownic Ransomware is a beast. Many suspicious strings were shown right off the bat, giving us clues as to what this malware is capable of. The author was very strategic in writing this malware, as almost nothing of value was shown to us in PEview or IDA. It was very intentional that the debug file referenced in strings was not provided, which likely would have aided my analysis in IDA. The partial encryption shown when initially running the malware was indicative of possible VM detection, which likely could have been found using x64dbg had the first anti-debugging method been bypassed.

For normal users, this ransomware can be difficult to deal with. All files could potentially be encrypted with no clear way to decrypt them. Even if there was a way to pay to decrypt the files (which there didn’t appear to be one), it is not recommended since files may still be left encrypted even after payment.

There is not a clear way to remove the malware either, since the malware cannot be closed out of or killed with task manager due to the malware’s ability to disable it. Clicking “Quit” on the first pop-up window will kill the malware, but it will still leave files encrypted, so this method is not recommended. The best way to remove the malware and restore files is to wipe the disk clean and have a recent backup to revert back to.

# Additional Notes and Future Analysis

I was originally running the malware on a netsec VM, but all of my screenshots from this analysis were lost. Instead, I ran the malware on a VM on my host machine. The one big difference that I found was on the netsec VM, the malware *actually encrypted* all of the files on disk at the time the malware was first executed: PDFs would not open, executables would not run (unless they were running prior to the malware execution), and text files had their contents replaced with encrypted text. The reason for the difference is not clear and is something to investigate further.

In the future, I would like to further investigate this malware when the tools become available, as there is a lot to unpack with this particular specimen. I would especially like to continue my analysis in x64dbg and see what other protections this malware has in place to prevent debugging and reversing.

# Resources

Malware download: [Clownic Ransomware](https://samples.vx-underground.org/samples/Families/Clownic%20Ransomware/)

VirusTotal Scan: [Clownic Scan](https://www.virustotal.com/gui/file/880823dd9df0ca6047cd829a1031e8a167ccec0629fdeac40a097dd555debf7c/community)