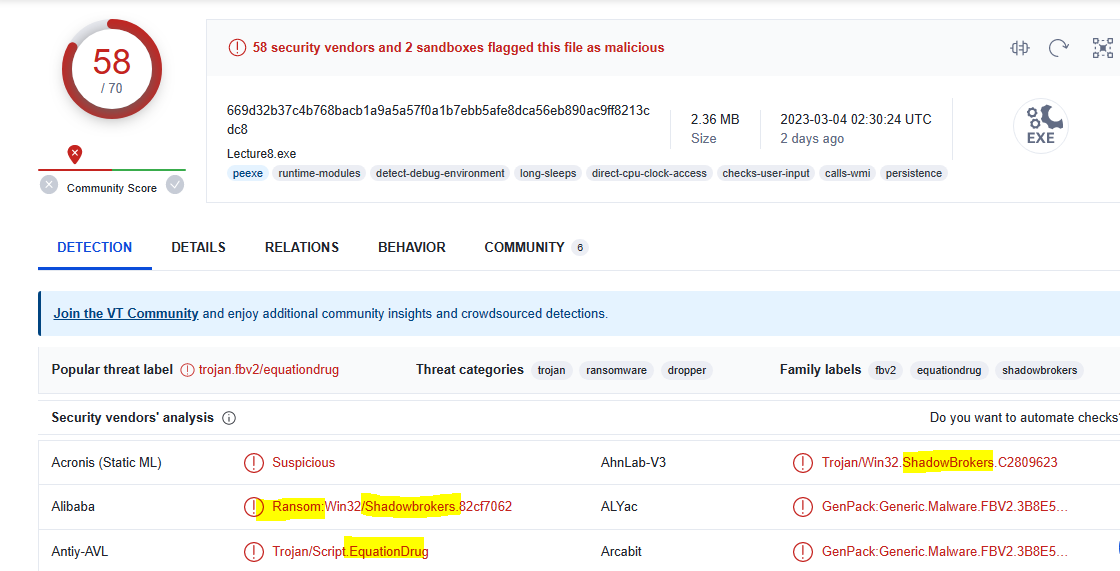
Answer the following questions using Lecture8.exe:

Answer the following questions:

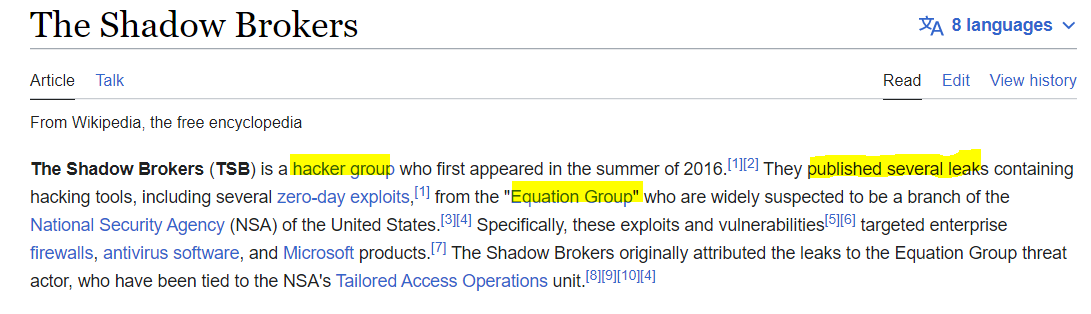
**Upload the file to**[**http://www.VirusTotal.com/**](http://www.virustotal.com/)**.**

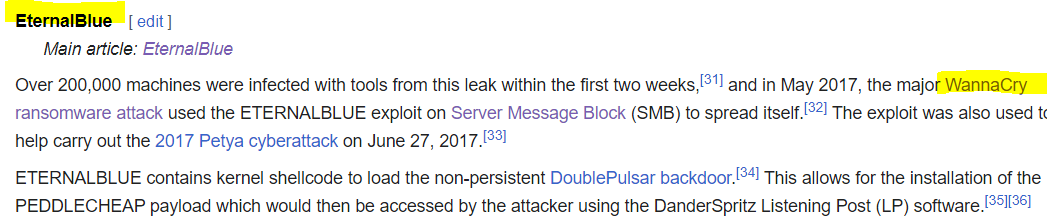
**Does the file match any existing antivirus signatures?** **What is this file known for?**

Yes, the file matches 58 existing security vendor antivirus signatures. There are a couple of notable names as to what this file is known for according to the security vendor analysis. We see unique names such as “Shadowbrokers” and “EquationDrug”. We also see indications that this file is a trojan of the ransomware variety.



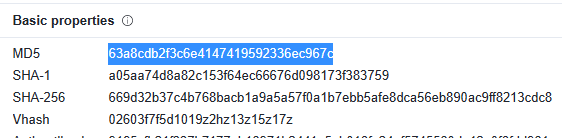
Conducting a Google search on the term “Shadowbrokers”, the first result is a Wikipedia page that defines The Shadow Brokers as a hacker group that publishes leaks containing hacking tools from the “Equation Group”.



The Equation Group is suspected, but not confirmed, to be a branch of the National Security Agency, so it can be interpreted that the hacking tools they publish are extremely powerful. We saw in last week’s discussion on the WannaCry malware that the EternalBlue exploit was created by the NSA and used by WannaCry. Further down in the Wiki, we see that EternalBlue was leaked by The Shadow Brokers and used in the WannaCry malware. This provides substance to the claim that The Equation Group is a division of the NSA.

**What is the hash of the file?**

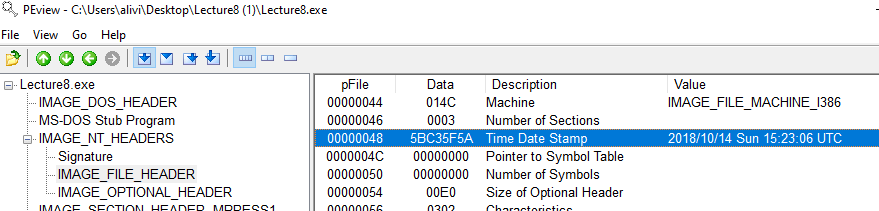
The MD5 hash of this file is 63a8cdb2f3c6e4147419592336ec967c.



**Using the tools, we discussed in lectures 1 and 2, answer the below.**

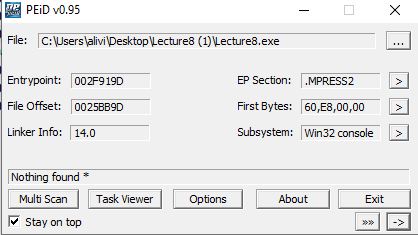
**When was this file compiled?**

When opened in PEview, we look under IMAGE\_NT\_HEADERS > IMAGE\_FILE\_HEADER and see a Time Date Stamp of Sunday 14 October 2018 at 15:23:06 UTC.

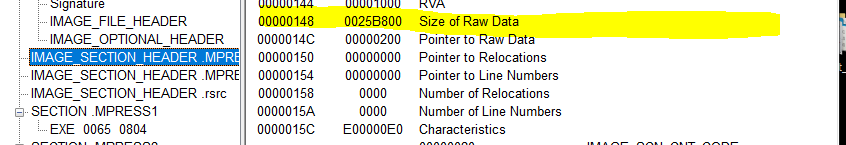


**Are there any indications that this file is packed or obfuscated? If so, what are the indicators?**

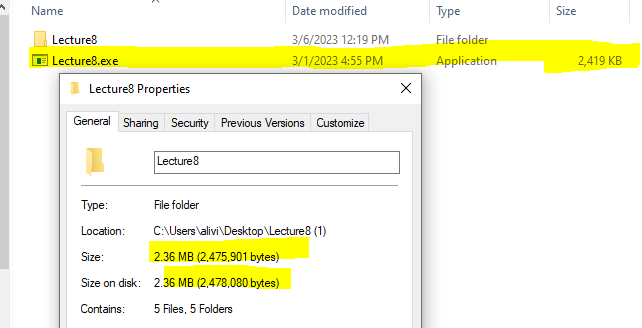
Packing: Conducting a hardcore scan within PEiD (click the ‘->’ button on the lower right and select “hardcore scan”) shows that nothing was found in the way of packing.



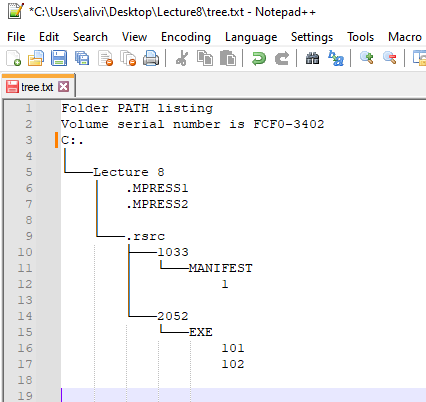
Packing: However, we also see in PEview that in IMAGE\_SECTION\_HEADER sections, there isn’t any information regarding the size of raw data. An example from one of the three sections is given below.



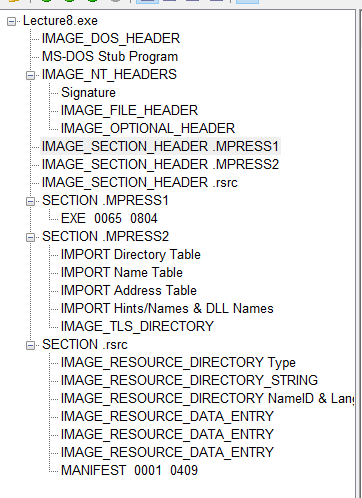
Packing: We also see that the file is rather large for a piece of malware at 2,419KB in the detailed view of file explorer, indicating that it hasn’t been packed. We also see that when we extract the file using UniExtract, the folder containing the files roughly match the size shown in file explorer.



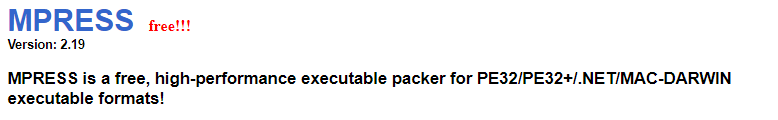
Packing: An added note with UniExtract: Within the folder that the UniExtract produced, I used Shift+Right Click and then clicked “Open PowerShell window here”. I then typed “tree /f > tree.txt” to get the directory and file tree. We see five total files. The .MPRESS files have file extensions of the same names and the numerically-labeled files simply of a file type of “file”.



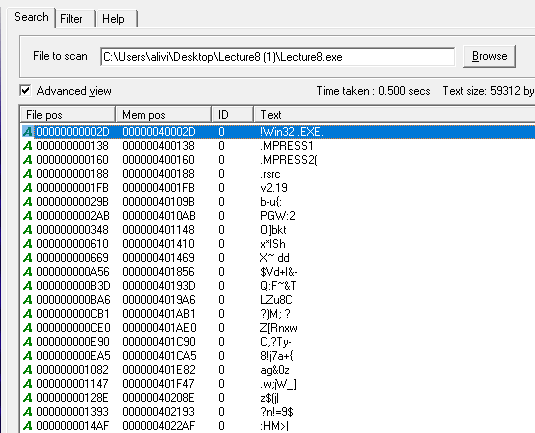
Packing: However, we notice that within PEview, there are not any .text portions to review and only the aforementioned “section” portions. This is likely indicative of some magnitude of packing. The other indication of packing is that we were able to extract even more files using the UniExtract tool rather than strictly seeing the Lecture8.exe file.



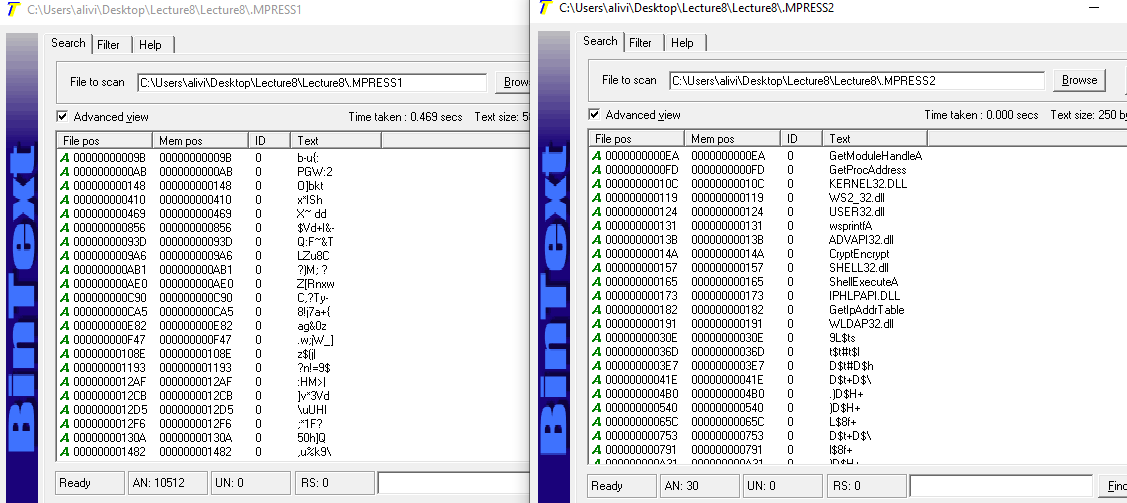
Packing: Upon doing a search of the .MPRESS file extension, we see that it is a result of file packing and commonly used with PE files.



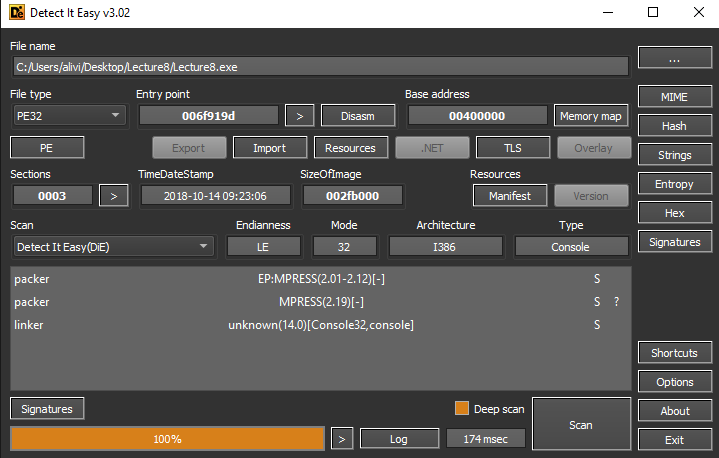
Obfuscation: Opening Lecture8.exe within BinText, we immediately see the .MPRESS1 and .MPRESS2 files identified within PEview and the files extracted from UniExtract. There are then multiple lines of code that appear to be random bytes of information. We also see the string “!Win32.EXE.” which indicates this might be the delivery process native to the machine for the file. The rest of the BinText output shows much of the same random data strings save for two XML calls and a note from Professor Galde.



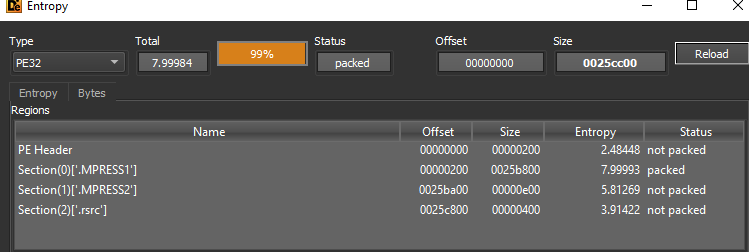
Obfuscation: Using BinText on .MPRESS1 and .MPRESS2, there is a lesser amount of obfuscation on .MPRESS2 and shows some imports. .MPRESS1 shows random data strings like what was observed in the BinText opening of Lecture8.exe.



Packing/Obfuscation Conclusion: Based on these findings, the file can be concluded as being packed with the MPRESS program in addition to being clearly obfuscated. PEiD did not detect any packing algorithms for Lecture8.exe, but the “Detect it Easy” program detected MPRESS v2.19 as the packing algorithm.

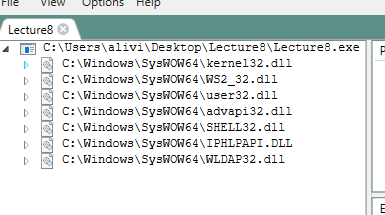


Specifically, it shows that the file .MPRESS1 was the only file that is packed when clicking on the “Entropy” button.

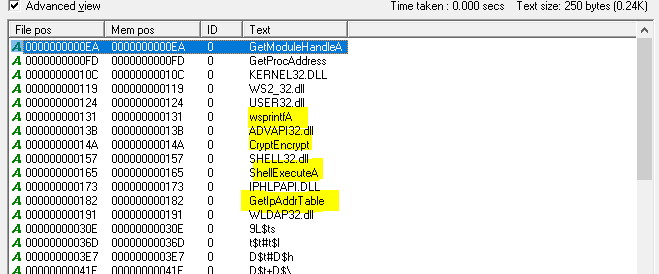


**Do any imports hint at what this malware does? If so, which imports are they?**

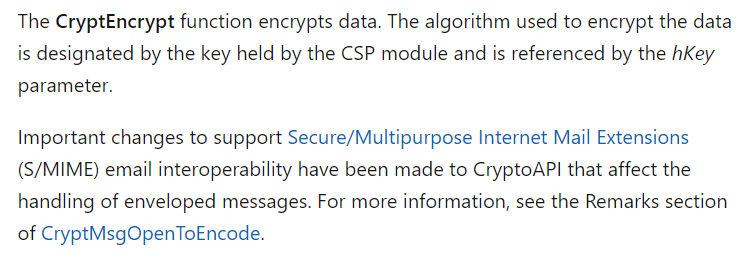
For Lecture8.exe, Dependencies shows seven .dll imports of kernel32, ws2\_32, user32, advapi32, shell32, iphlpapi, and wldap32. We know that kernel32 will give the file access to hardware functions and the kernel and user32 will give it access to the user interface. Advapi32.dll indicates that core Windows components will be altered, such as the Service Manager and Registry. Ws2\_32.dll contains networking functions and indicates that the file will perform some of those functions. Iphlpapi.dll is the Windows IP Helper API contains functions for use with the Windows Shell and other IP-related tasks. Wldap32.dll is the LDAP API dynamic linked library. LDAP stands for “Lightweight Directory Access Protocol” and uses port 389 over TCP or UDP. If LDAP is used in conjunction with TLS/SSL for secure connections, it will use port 636.



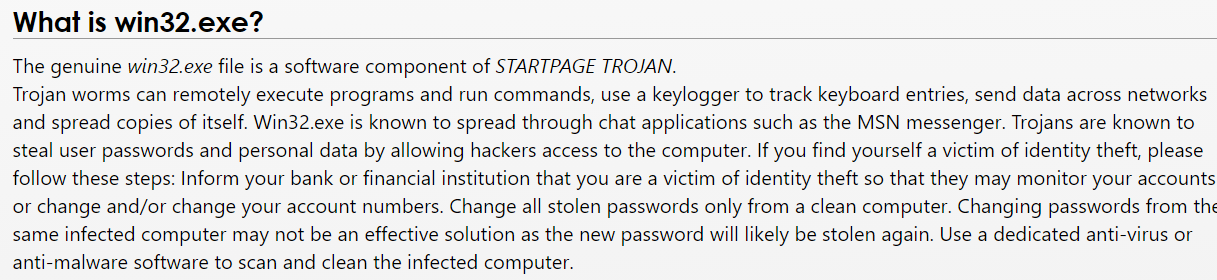
Within BinText, we can see some imports of the same dynamic linked libraries within .MPRESS2 as we did in Lecture8.exe (when examined in dependencies). We see a few extra imports here, highlighted in yellow. We see [wsprinfA](https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-wsprintfa), a function of winuser.h, which writes formatted data to the specified buffer. [ShellExecuteA](https://learn.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecutea) is a function of shellapi.h which will perform an operation on a specified file. Then there is [GetIPAddrTable](https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getipaddrtable), a function of aphlpapi.h, which will retrieve an IPv4 address mapping table. These functions will be important to look for within IDA Pro to see what exactly they are going to do.



The most interesting import within the above screenshot is [CryptEncrypt](https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptencrypt), a function of wincrypt.h, which encrypts data. Reviewing the documentation on this function, we see that “the algorithm used to encrypt the data is designated by the key held by the CSP module and is referenced by the hKey parameter.” When we examine the call to CryptEncrypt within IDA Pro, we can check the parameters passed into it to retrieve the hKey value and potentially use the function [CryptMsgOpenToEncode](https://learn.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptmsgopentoencode) to decrypt the data. This function was referenced in the CryptEncrypt documentation.



The other import of note was found when examining the BinText view of Lecture8.exe and seeing the executable import of “!Win32.EXE”. According to file.net, this executable that is a software component of “STARTPAGE TROJAN”.



**Are there other files or host-based indicators you could look for on infected systems?**

Based off the naming convention of the file from the security vendors from VirusTotal identifying it as potential ransomware, there will likely be popup windows that inform a user to pay a ransom in order to decrypt their files. If a user does not compress their files with MPRESS, it is likely that any file extensions of “.MPRESS” will be indicative of infection. Additionally, the existence of win32.exe within a user’s system is also indicative of malware infection based off of the file.net screenshot above.

**BEFORE you run this malware, would you consider this file malware based on your findings? "Exclude your virus total findings."**

Yes. The key factor in determining this is the win32.exe string identified in the strings analysis and cross-referencing it with known applications that go by that name. As stated before, file.net identified it as a software component of “STARTPAGE TROJAN.”

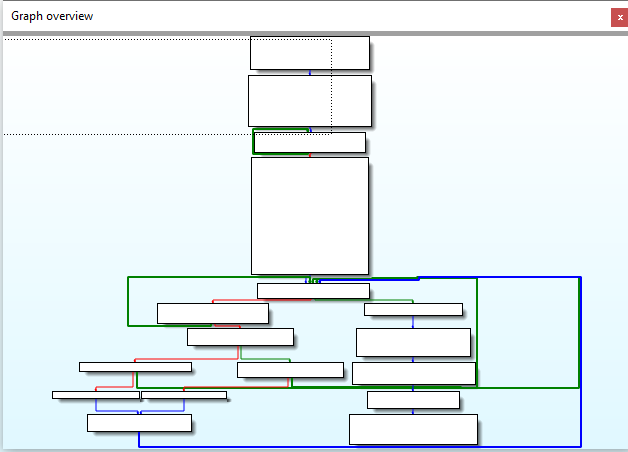
We can further deduce the behavior of the malware by looking at the imports. We see calls to encryption functions, shell creation, and other network-related imports. It is reasonable to assume that this malware will encrypt files on a host machine and allow the attacker to establish an encrypted connection for payment like we saw with the WannaCry malware. The shell could also be used as a backdoor to the infected system and allow attackers to randomly encrypt files to further extort payment from a victim.

* \****ONLY IN THE VM\****

**TURN OFF NETWORKING!!!!!**

**Using the tools, we discussed in lecture 6, answer the questions below and provide screenshots.**

**Provide a screenshot of the graph view of the program.**



**Provide a screenshot of the navigator bar (the color-coded bar showing library functions, regular functions, code, data, etc.)**

Part 1:



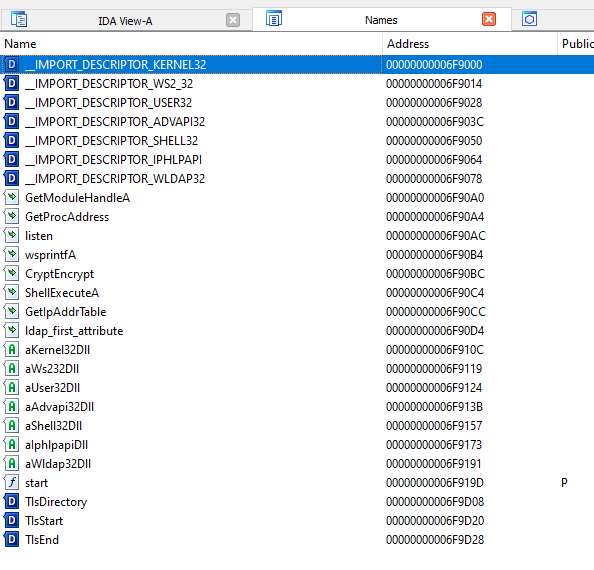
Part 2:



**Open the Names Window and view every function, Library, Code, String, Dara, and Linked Function. Provide an analysis of which of these are "interesting."**

Overview:

Viewing the Names Window, we see the defined data matching the string imports identified in the .MPRESS2 file and the Dependencies output of Lecture8.exe. We also see some functions that were previously identified, such as “CryptEncrypt”, “ShellExecuteA”, etc. We also notice that the “start” function is the only “Public” portion of this code and is what was previously captured within the graph overview.



As previously mentioned, we noticed within the PEview that there wasn’t a .text section to analyze. This is confirmed when examining the contents of each value in the Names Window. Beginning with the name “aKernel32Dll”, we don’t enter a .text section but rather a .MPRESS section (every name prior to aKernel32Dll is in the .data section).

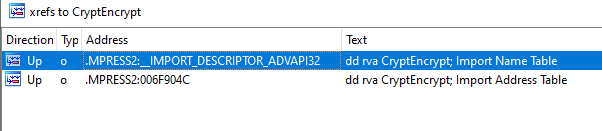


Function Definitions:

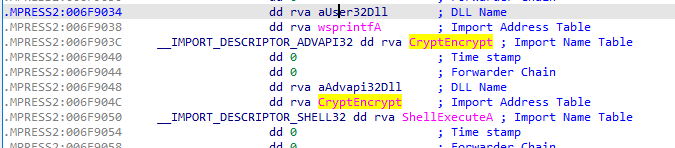
Examining the names of some key functions identified earlier, we notice that definitions for them are consistent with their documentation and can see the parameters for them. This information will be critical, especially for the function call of CryptEncrypt and figuring out what the hKey is in case files do get encrypted.



Looking at the cross references for each of these functions in the screenshot above, they both have two in the xrefs window. We see an import descriptor and another address.

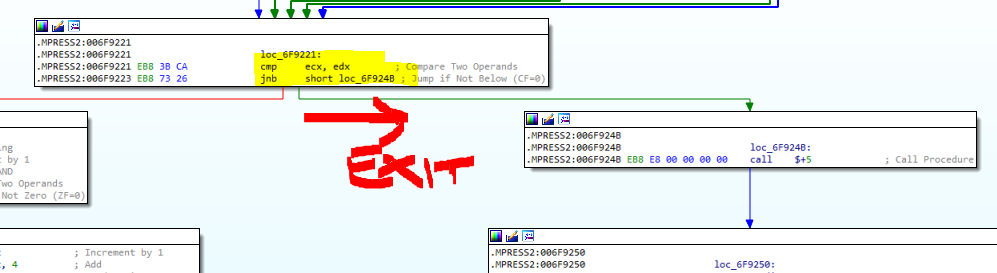


Although there are two cross-references for each function, they are both referenced within the same area within the \_IMPORT\_DESCRIPTOR portion. My best guess from these examples is that they are simply function import definitions. In a python syntax, it would read similar to “from advapi32 import CryptEncrypt”.

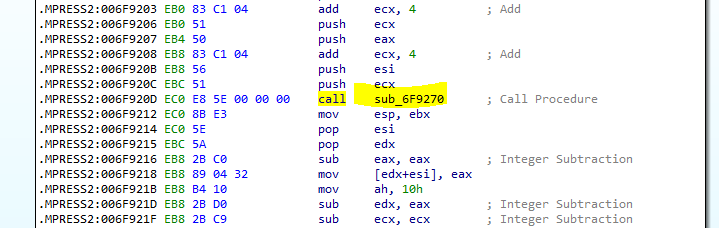


“Start” Function:

Examining the “start” function, we notice that all the code for this function occurs in the .MPRESS2 section. Interestingly, we don’t see any calls to the functions identified previously. We also notice there is an endless loop for this function from the graph overview and only one portion that prompts an exit with a “cmp ecx, edx” instruction that doesn’t set the change flag to 0.

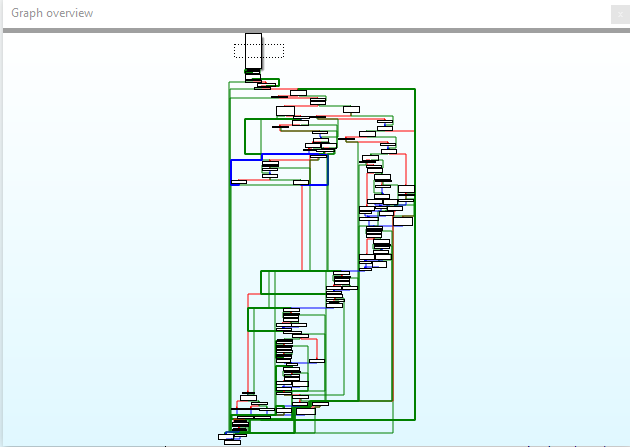


There are four “call” instructions within start, three of which call the procedure defined as “$+5” (see above screenshot). The other call is for subroutine “sub\_6F9270” which occurs in the third “block” of code (when viewed in the graph view) of the start function. I was able to navigate to sub\_6F9270 but not to $+5, indicating that the $+5 procedure is inaccessible or encrypted.

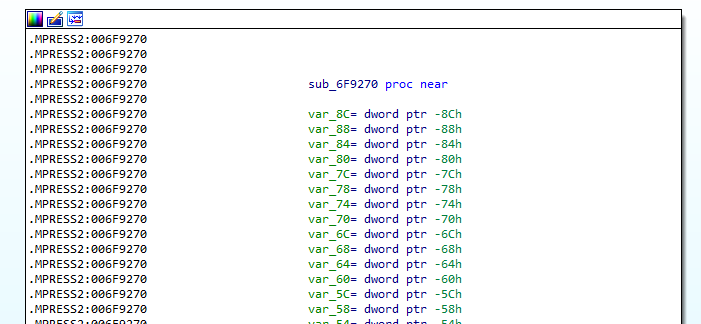


Sub\_6F9270:

Subroutine 6F9270 appears to be a much larger and complicated function than start.

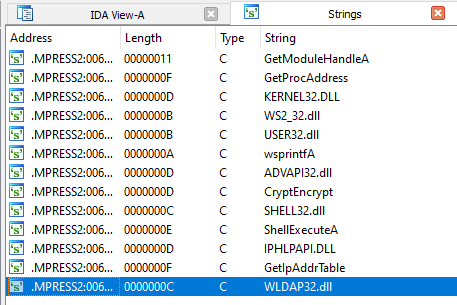


The local variables of this subroutine are in hexadecimal form and are called as such.



**Open the Strings window and provide an analysis of which ones are interesting and why.**

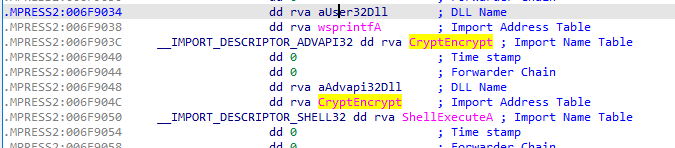
Within the Strings Window, we notice much of the same functions and .dll imports identified earlier in this analysis.



The strings are all within a segment of type “Pure Code” and each has a cross-reference.



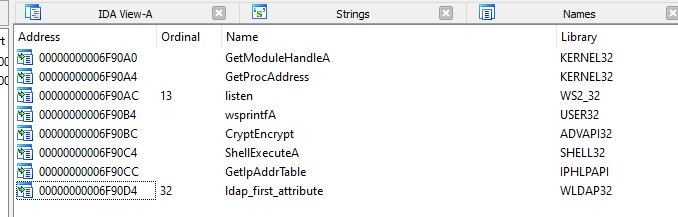
When following the cross-references, we get back to the same location identified within the “Function Definitions” subsection of the Names Window findings.



**Open the Imports and Exports windows and provide an analysis of your findings.**

Imports:

Within the Imports window, we see the same functions from the libraries identified earlier.

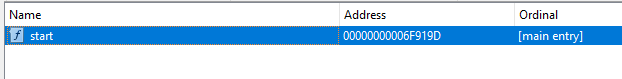


When following these imports, we get back to the same location identified within the “Function Definitions” subsection of the Names Window findings.



Exports:

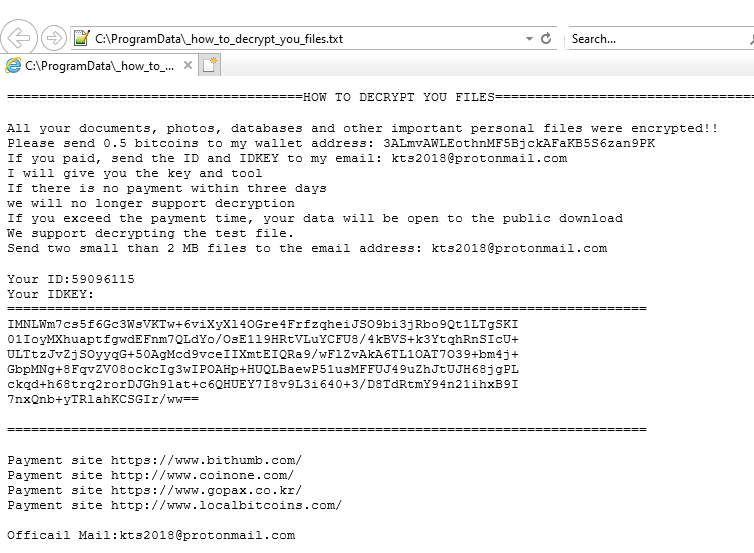
We only see the start function with the ordinal defined as the main entry point.



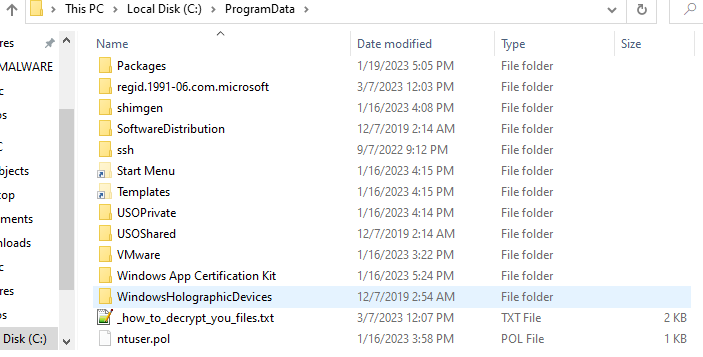
**RUNNING THE MALWARE**

**Observations**

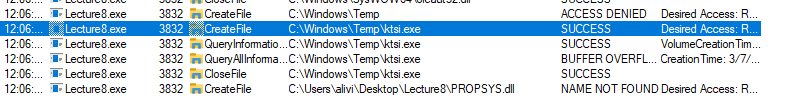
Immediately, an Internet Explorer window popped up that displayed a .txt file with a path to the C:\ProgramData directory. This is the ransom note that we anticipated. It displays an email address, several websites, and a bitcoin wallet address.



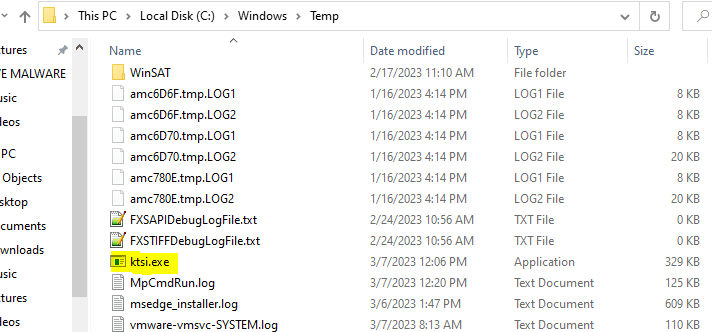
This file was confirmed to reside in the specified path.



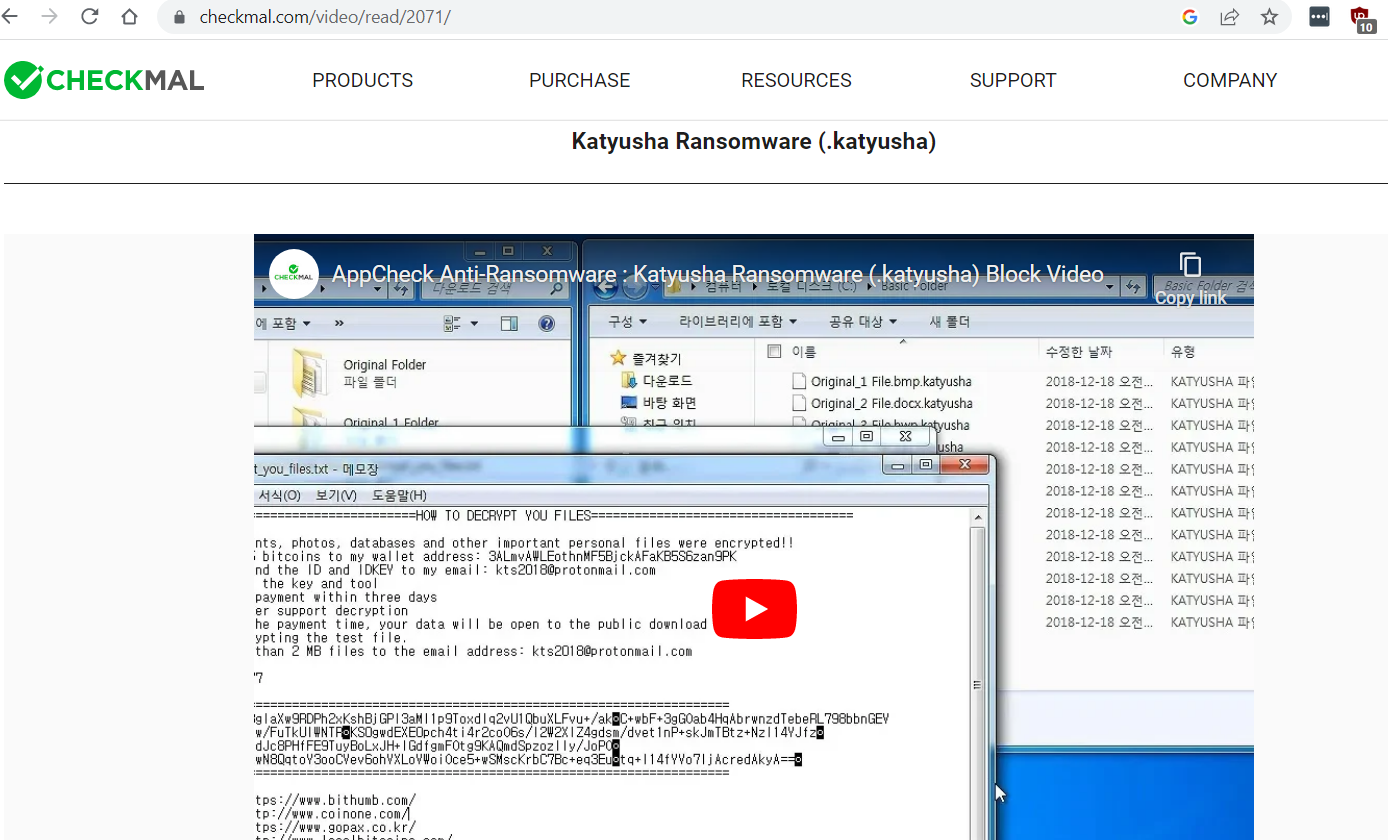
When examining procmon for any suspicious file activity, the first indication was a reference to “ktsi.exe” within the Windows\Temp folder.



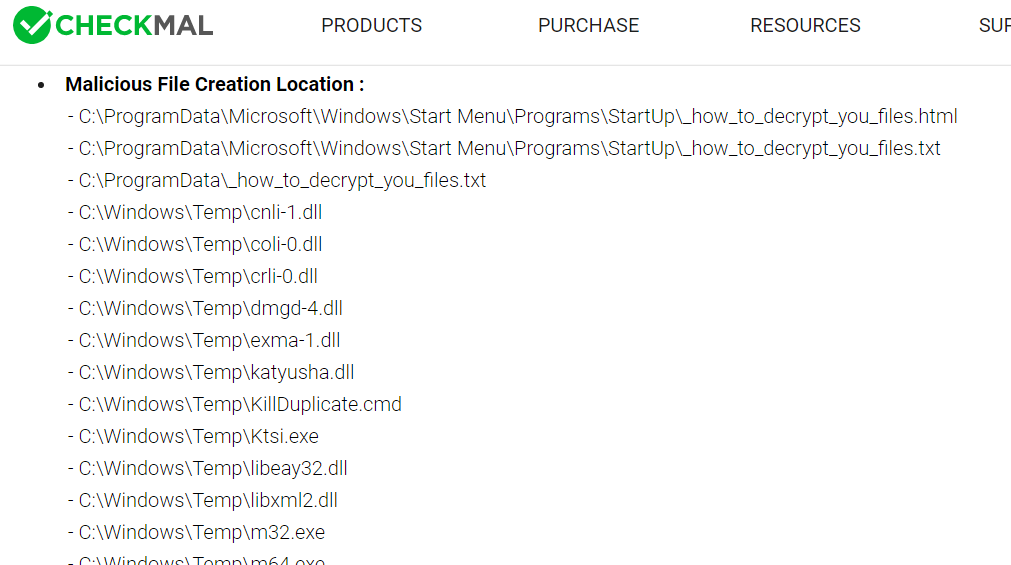
Ktsi.exe was confirmed to reside in that folder.



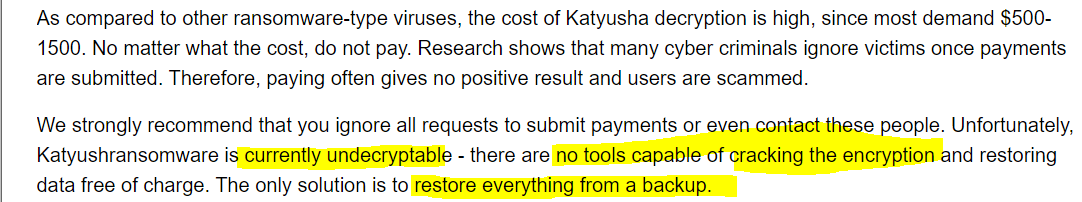
A Google search of this executable stated that it is a malicious file and is the “Katyusha” Ransomware. The website shows an identical ransom message to the one that popped up on my screen.



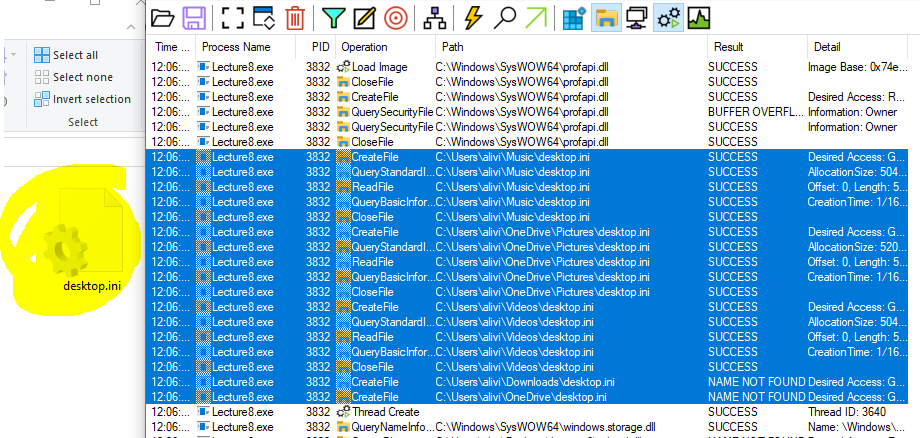
The website found from the Google search also shows multiple creations of malicious files within the Windows\Temp folder (the list is extensive).



The ransomware operates by encrypting files with the extension “.katyusha” and is stated to be not decryptable with the only solution being to restore everything from a backup (article from 2021).



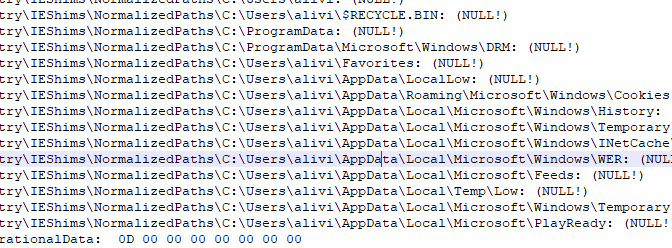
Procmon also showed creation of multiple files of “desktop.ini” within the “Users” directory.



Process Explorer showed Lecture8.exe with a child process of conhost.exe.

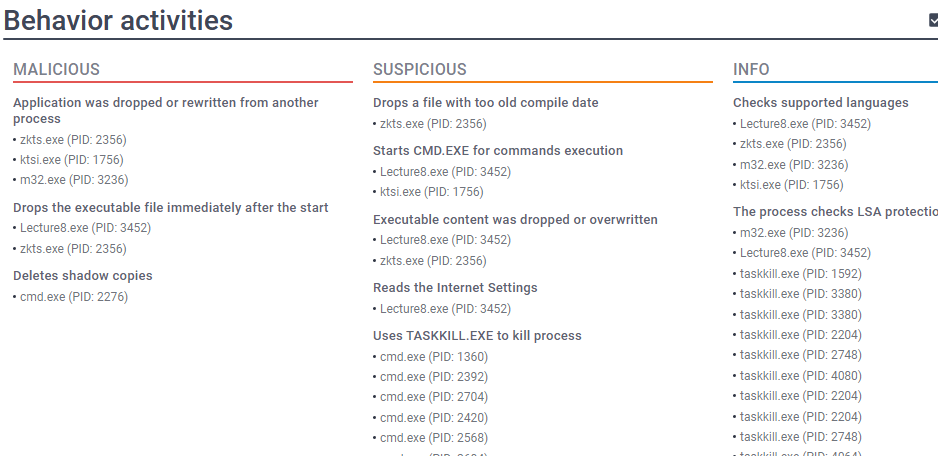


Regshot captured 570 changes within the registry after the malware was run. There was an interesting change that occurred within an “IEShims” key path, which indicates modification to internet explorer settings and potentially inserting shims for the malware to operate.

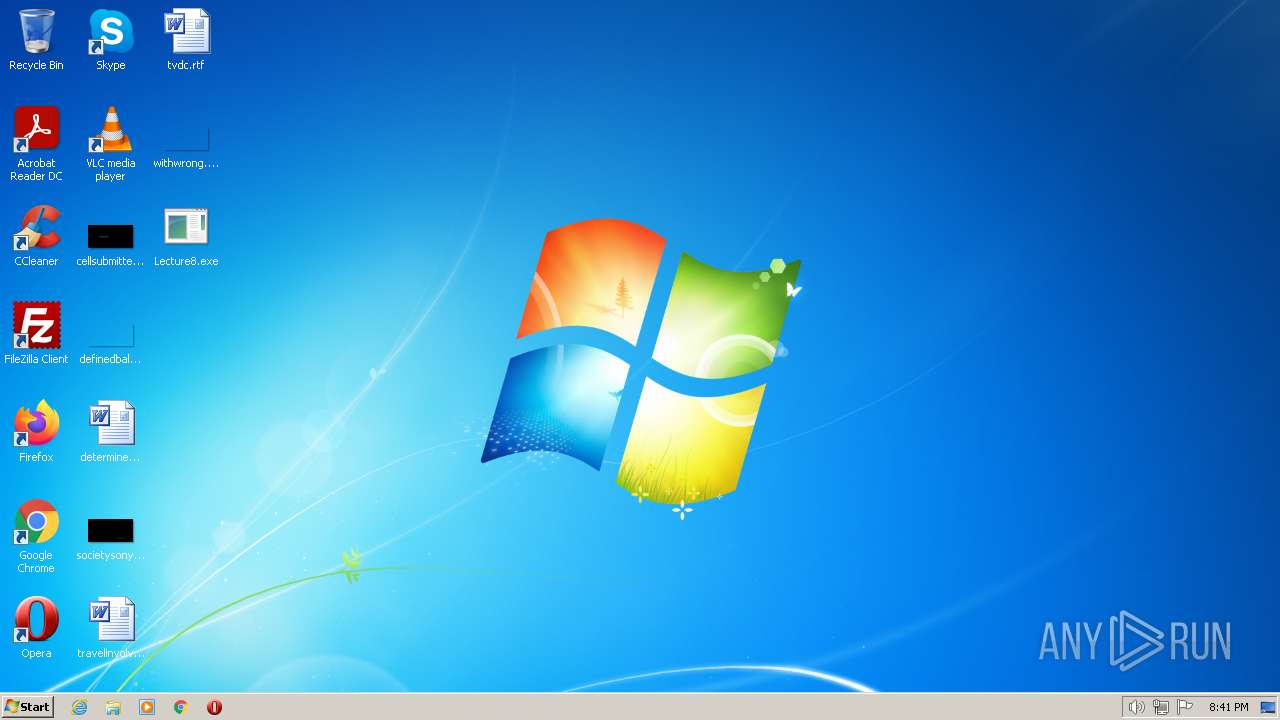


What is most interesting is that this malware did not encrypt any files. I did a search through the entire drive for “Katyusha” and did not find a single hit.

When the malware was ran on app.any.run, there was plenty of malicious activity detected. Some other .exe files that were run were zkts.exe and m32.exe in addition to ktsi.exe. It also showed activity to delete shadow copies of cmd.exe and a lot of taskkill.exe instances.

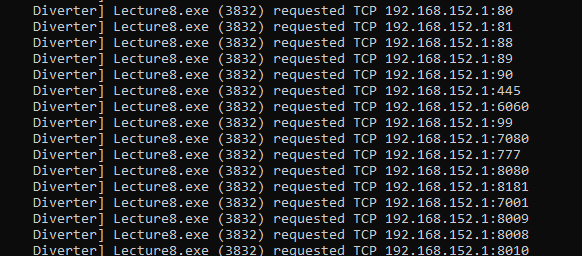


Since app.any.run has sample word documents, I expected that they were going to be encrypted with the .katyusha file extension, but the desktop did not show anything of the sort.

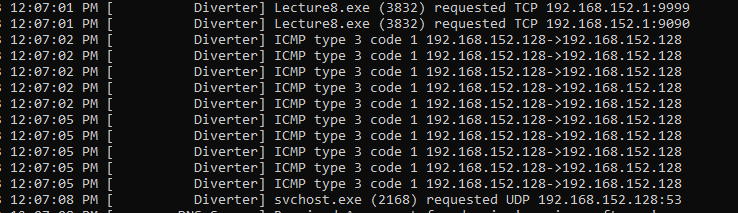


**What network-based indicators could be used to find this malware on infected machines?**

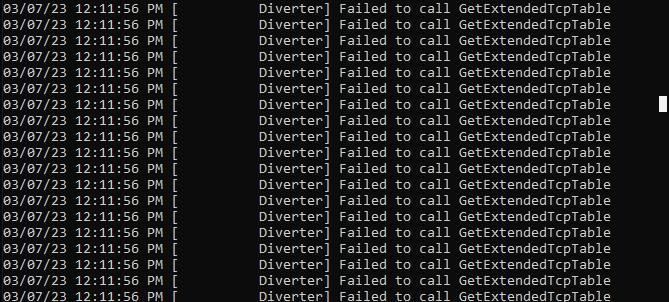
Fakenet immediately showed a lot of activity, requesting TCP data from the machine’s local IP address over numerous ports. This is most likely a result of the GetIPAddrTable function identified in the static analysis portion of this report.



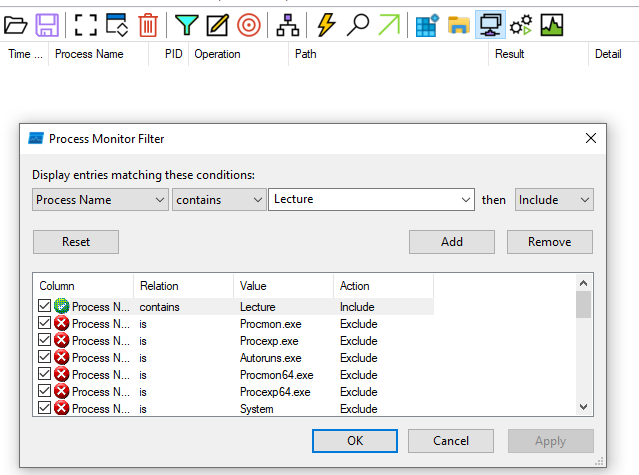
These TCP requests were followed by ICMP Type 3 requests.



There were also a lot of messages that stated “Failed to call GetExtendedTcpTable” in Fakenet.



Interestly, procmon did not show any network activity from Lecture8.exe, nor did it show any network activity captured at all.



All of these network-based indicators suggest that there are no outbound connections by the malware, but instead is using the protocols imported by the dynamic linked libraries to traverse directories and find IP routing tables.

It should be noted that app.any.run did not detect any DNS requests nor outbound connections to suspicious IP addresses.

**What would you guess is the purpose of this file?**

I would guess the purpose of this file is to obfuscate and pack its intentions by using the MPRESS packing software. It then uses the obfuscated code within the MPRESS1 file that contains the exports such as ktsi.exe and install them on the machine it infects. I also believe that most of the useful code resides within the MPRESS1 file since much of the assembly code within the MPRESS2 file references the MPRESS1 section. It would therefore be impossible for someone to deduce the precise functionality by simply examining the MPRESS2 assembly code without having fully unpacking or decrypting the MPRESS1 file.

I attempted to find an MPRESS unpacker to further analyze these files, but the process became too long and arduous to make the “juice worth the squeeze”.

The research of the win32.exe file found in the strings analysis immediately indicated that this file was malware. After the malware was run, the ktsi.exe file pointed to the specific Katyusha Malware which is helpful in threat attribution and signature generation. Based on the research of Katyusha, I expected that the malware would have encrypted my files with a .katyusha file extension (much like the .WNCRY extension), but found that it did not. Perhaps it is only targeting specific file types such as .docx but since I do not have Office installed on my VM, I could not analyze this functionality.

The imports of networking functions from networking dynamic linked libraries but not detecting any outbound network activity is interesting. I believe that instead of using standard directory traversal functions like we saw in WannaCry, it is having the host machine connect to itself and using the LDAP protocol for directory traversal. Since the file showed continuous self-connecting network activity, it is possible that it searches the entire directory tree of the machine to encrypt any new files that appear.

To further analyze the malware and the purpose of its network-related functions, it would be a good idea to run it with an active network connection to see what additional, if any, network connections might be made.

**What programs do you see open, what is the malware trying to do?**

See observations above.

**Now, how often would you be able to run this VM, excluding snapshots?**

Since nothing was observed as being encrypted, I could still run this VM without reverting to a previous snapshot. All the programs still worked, but further investigation in regards to more program functionality would need to be done.

**What would your recommendation be to Management? Do we need to stop generating revenue and cleaning, or can we go on and clean as we go?**

If files were actually encrypted, we would need to stop generating revenue and restore the infected machine from a known good backup. This is because the research done on the Katyusha malware shows no good method to decrypt the files.

**Can you clean the system, and if so, how would you do it?**

I would not be able to clean the system if the files were encrypted since the files cannot be decrypted.

**Go to two other student's posts and observe their findings. Post if you agree or disagree with the results.**

My responses:

<https://d2l.arizona.edu/d2l/le/1243099/discussions/threads/9678426/View>

<https://d2l.arizona.edu/d2l/le/1243099/discussions/threads/9679274/View>

**Did they find something using a new technique, and if so, would you use this next time?**

Christian showed me the importance of using Wireshark so I can analyze the packets and get more details on IP addresses. Babek showed evidence of this being network-spreading ransomware and also taught me the meaning of ordinals and how to use them in future analyses.

**From a business perspective, if you were both being paid at the same rate. Would your analysis be more cost-productive and achieve the same results?**

Once again, I think I spent too much time on this and was way too wordy with my explanations. I believe that my analysis was not as cost-effective as the others I have reviewed for this assignment, but we did achieve most of the same results, especially identifying this as Katyusha ransomware.

**Suppose you were working on this malware to see if this could be allowed in your organization. Did your analysis provide enough detail to make this determination?**

Yes. I was able to make the determination that this was malware just by a simple strings analysis and finding the win32.exe file.