

Bailey Powers

Reverse Engineering Fundamentals

CSEC 202 Section 01

4/15/2022

-Phobos Ransomware Analysis-

Table of Contents

Introduction	3
Sandbox Setup	3
Basic Static Analysis	4
VirusTotal	4-6
Strings	6-7
PEiD	8
PEview	9-10
Resource Hacker	10
Basic Dynamic Analysis	11
Running The Ransomware	11-13
Process Monitor	13-15
Process Explorer	16-17
RegShot	17-19
Wireshark	19
Advanced Dynamic Analysis	20
IDA Pro	20-22
Advanced Static Analysis	22
X32dbg	22-24
Conclusory Analysis	25
Potential Dangers	25
Mitigation	25
Citations	26

Introduction:

Phobos Ransomware, first detected in December 2017, was crafted by a ransomware gang and used to target smaller businesses and hospitals. Ransomware targeting a hospital could be extremely dangerous to patients as many hospitals run off their systems. Smaller businesses might not be able to keep up with the price and could go bankrupt. This method is unlike many other ransomware gangs, mainly targeting larger enterprises. This ransomware worked differently from others as it did not ask for a specific amount of money outright, but the victims were meant to send an email to the given address and discuss the price point to get all the files unencrypted. On average, the amount that was discussed for the ransom was around \$27,050, which looks pretty high for a smaller business, but prices could range. Phobos was one of the most prominent ransoms used throughout 2019 and 2020, with newer variants and offspring found as recently as April 2021.

Phobos, along with its relative ransomware, Dharma, prays on poorly secured RDP connections and allows itself onto any of the other systems in the network using this strategy. All of the files that it can get its hands on are encrypted and have “.phobos” attached to the end. The ransomware also deletes any backups or shadow files it comes across, making a recovery take much longer.

Testing Environment:

A copy of the ransomware was acquired from the site VX-Underground for this project. VX-Underground is a website that contains millions of malware samples that can be dissected for malware analysis such as this. The malware sample comes in a password-protected 7zip file with the password being "infected". After unzipping the file adding ".exe" to the container file will allow it to be run on the system. The sandbox environment used to do the analysis was done in a Windows 10 virtual machine running on a VMware workstation. Inside the Windows 10 virtual machine, the network card was disconnected so that there could be no communication from the virtual machine to the host machine. Along with Windows Defender was turned off to make it easier to run the malware and make sure that it was running to its full effect to get a proper analysis. By taking a snapshot before running each program to test the malware, the virtual machine could be reverted to a previous state if all of the files were encrypted, and the VM would not have to be reset every time.

Basic Static Analysis:

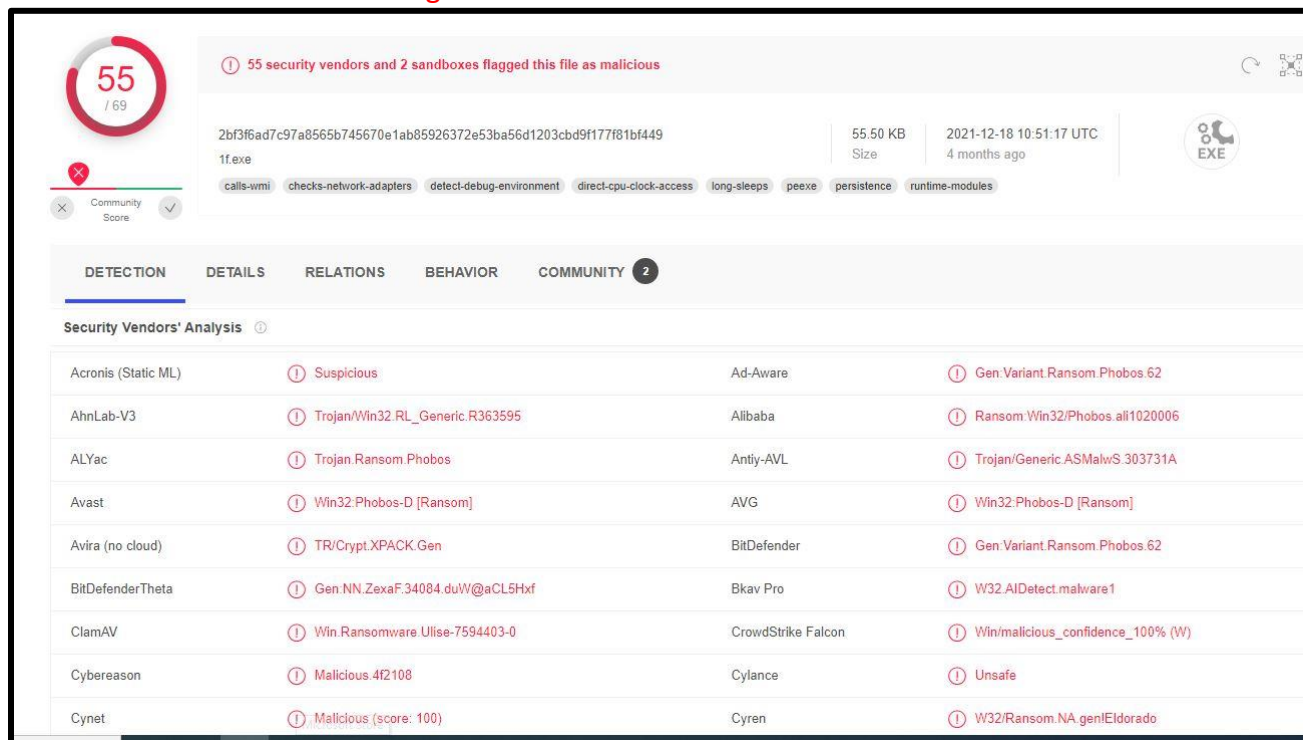
The main concept of basic static analysis is to find everything about a program that you can without running it or looking at the source code/assembly. This would mean only things that could be found on the given file without editing or changing anything. Anything being strings, DLLs, imported functions, or any other resource can be seen in the file. When dealing with Phobos, five different tools will be used to try to get information.

Virus Total:

Virus total can be an essential tool for basic static analysis. When the file is uploaded to the site, it is tested against around 70 different antivirus tools to see how they identify the file. This site also retains memory if it has seen a sample before. This means that any information people have about the sample can be left as a comment for others to see. This site also allows the user to see the creation date, attached DLLs, IP address, and some attached functions.

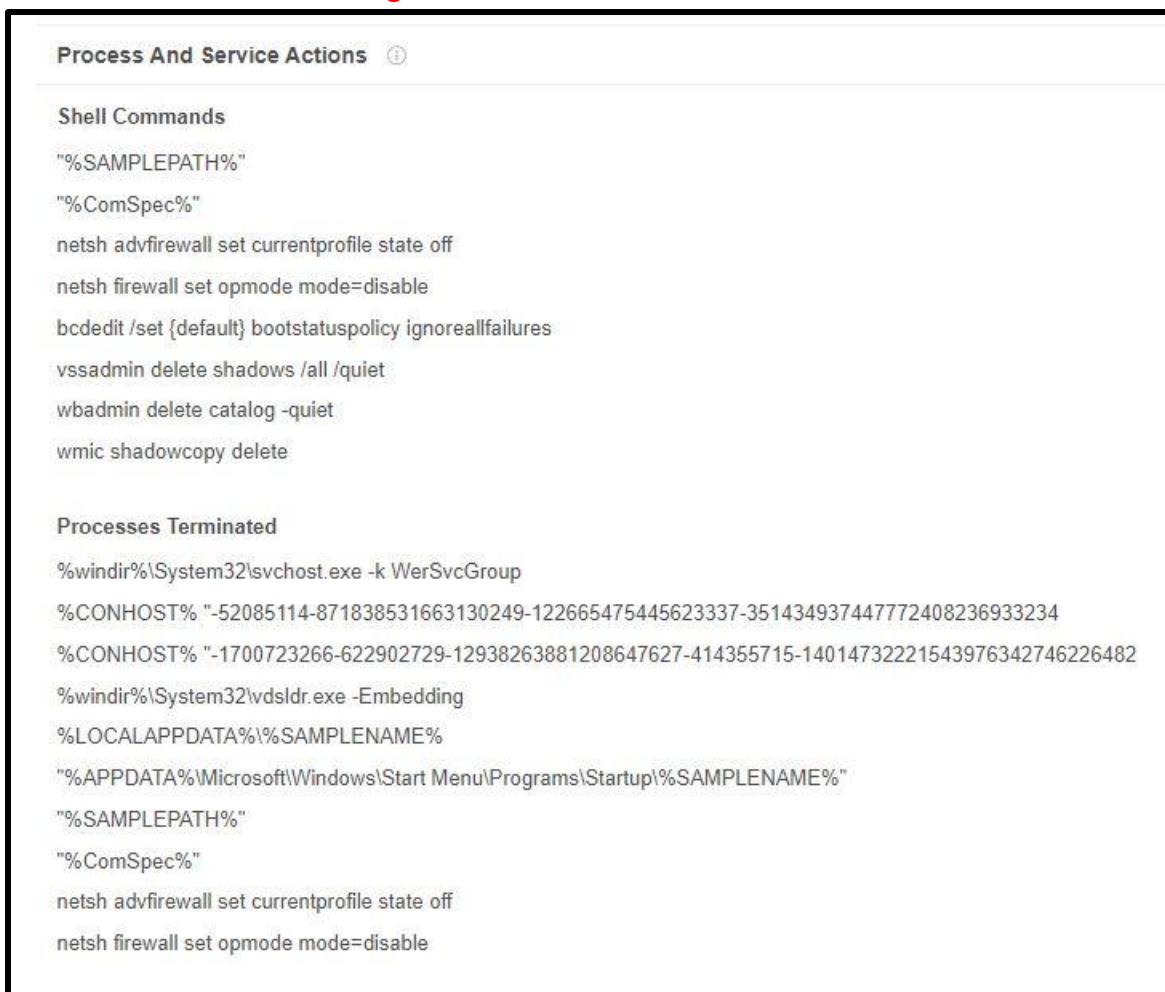
We are met with the screen below when running Phobos through the virus total. This means that 55 of the 69 antivirus tools detected the given file as malware and could be an extreme risk. With this, we can see what they are classifying it as, and most of the tools are correct in recognizing it as the Phobos ransomware. Along with all of these, we can see some of the associated tags, such as persistence, detect-debug-environment, and checks-network-adapters. All of which could be pointing to signs that this will be a tricky malware to reverse as it can tell if it is being run inside of a sandbox environment and might change its effect because of that.

Figure 1: VirusTotal antivirus check



Something interesting that I did not know that the virus total did is that it also allows you to see some of the functions present within the file. From this, we can see that some of the shell commands that it runs are “netsh advfirewall set currentprofile state off,” which means that one of the first things that the ransomware did was turn the firewall off so it could potentially connect to something that is blocked by the firewall. Another thing that I found interesting within these is the command “vssadmin delete shadows /all /quiet” and “wbadmin delete catalog -quiet,” which are two commands that first deletes all of the shadow copies within the system and then deletes the backup catalog from both commands respectively. Interestingly, both of these commands are that they are both being run with a quiet attribute so that the victim does not see either of these being done.

Figure 2: VirusTotal Processes and Actions



Virus total also lets you see what IP addresses are connected to the malware. From this, we can see that the most prominent IP address in the file is “20.190.159.134” which connects back to a location in Ireland if looked up in a geolocator. All the other IPs listed also connect back to Ireland, potentially giving us an idea of where the malware originated.

Figure 2: VirtusTotal Connected IPs

Contacted IP Addresses ⓘ			
IP	Detections	Autonomous System	Country
20.190.159.132	1 / 89	8075	IE
20.190.159.134	2 / 89	8075	IE
40.126.31.1	1 / 89	8075	IE
40.126.31.135	1 / 89	8075	IE
40.126.31.137	1 / 89	8075	IE
40.126.31.139	1 / 89	8075	IE
40.126.31.143	1 / 89	8075	IE
40.126.31.4	1 / 89	8075	IE

Strings:

Strings.exe is a popular tool that can be used to look through the Unicode within a given file. When the script comes across a string with three characters in a row, that is seen as important. This can be used to look through the given file to return information such as DLL names, used functions, or hidden metadata like IPs or passwords hidden within the file. Strings were used on the Phobos exe file for our basic static analysis.

When Strings is run on Phobos, a few things stand out and help us get a better idea of what the malware is doing. The first thing that I thought was interesting was all of the function calls using HTTP, as seen in Figure 3. This could mean that the ransomware is potentially reaching out for some extra functions or information. This could also mean sending some information from the host machine out to a specific IP address.

Figure 3: Interesting Strings

```
WinHttpRequestReceiveResponse
WinHttpRequestSendRequest
WinHttpRequestConnect
WinHttpRequestCloseHandle
WinHttpRequestOpen
WinHttpRequestOpenRequest
WINHTTP.dll
```

A few other strings that are important to see what the ransomware is doing are the ones that are found in figure 4. From the function calls below, we can see that it is using "LookupPrivilegeValueW," "AdjustTokenPrivileges," and "RegSetValueExW." From these, we then know that the malware is doing something with the registry. Specifically, it is editing one of the privileges of the account and changing different registry keys.

Figure 4: Other Interesting Strings



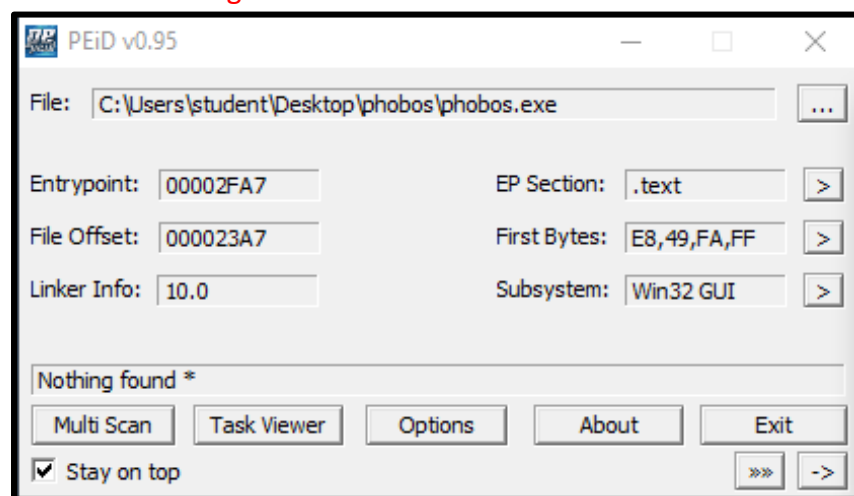
```
OpenProcessToken
GetTokenInformation
EqualSid
RegQueryValueExW
LookupPrivilegeValueW
LookupAccountSidW
DuplicateTokenEx
AllocateAndInitializeSid
FreeSid
RegOpenKeyExW
AdjustTokenPrivileges
RegCloseKey
RegSetValueExW
```

PEiD:

The next important tool I used to do basic static analysis is PEiD. This tool is used to determine whether malware is packed. This shows how the malware was compiled and what kind of system it was compiled for. Knowing this information is important as you will know how to proceed with the reversing.

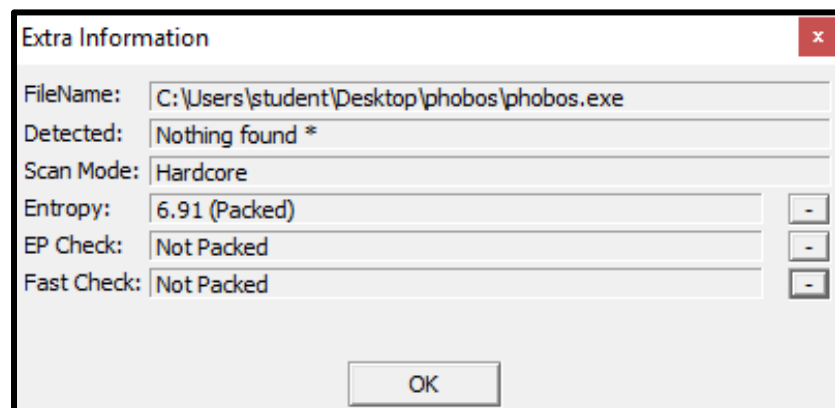
We get back some interesting information when running Phobos ransomware through PEiD (see figure 5). First, we see that the malware's subsystem was compiled for was Win32 GUI which will work on most windows systems. Other than that, the first few bytes are E8, 49, FA, and FF.

Figure 5: PEiD with Phobos malware



Next, by pressing the extra information button, we can see whether or not the malware is packed. Figure 6, through the entropy check, shows that the ransomware is 6.91 packed. The other two, EP Check and Fast check, are coming back as neither being packed. This is important because, knowing this, we know that the ransomware will be harder to debug.

Figure 6: Extra PEiD Information



PEview:

PEview can be a beneficial tool in being able to read the binary of the malware. PEview allows the user to look into different categories that it puts each part of the binary into and translates into human-readable text. Examples of this are pulling the creation date and compiler type out of the binary and displaying it in plaintext. Along with this, you can view all of the functions used just like the ones listed in strings, but in this case, they have all split into the sections of the DLLs that the individual functions come from.

When Phobos.exe is loaded into PEview, we are met with a few categories to look at, as seen in Figure 7. The first important place to look is in the IMAGE_NT_HEADERS dropdown. Inside here, you will find the translated hex into different values that could be important when looking at the malware. As seen in Figure 8, the file header is loaded with the values. First, we can see the date of creation of the malware, which claims that it was compiled on March 31st, 2020. This contradicts what we saw on Virus total, claiming that it was compiled much later. This could be a case of metadata manipulation, or it could just be the virus total going off different examples of the same sample. Another important point to look at here is the section labeled "0100 IMAGE_FILE_32BIT_MACHINE". This section means that the malware was compiled for 32-bit operating systems.

Other than those facts about the malware compilation, there is just the information we previously had about each of the DLLs and functions that are called during its use.

Figure 7: PEview with Phobos

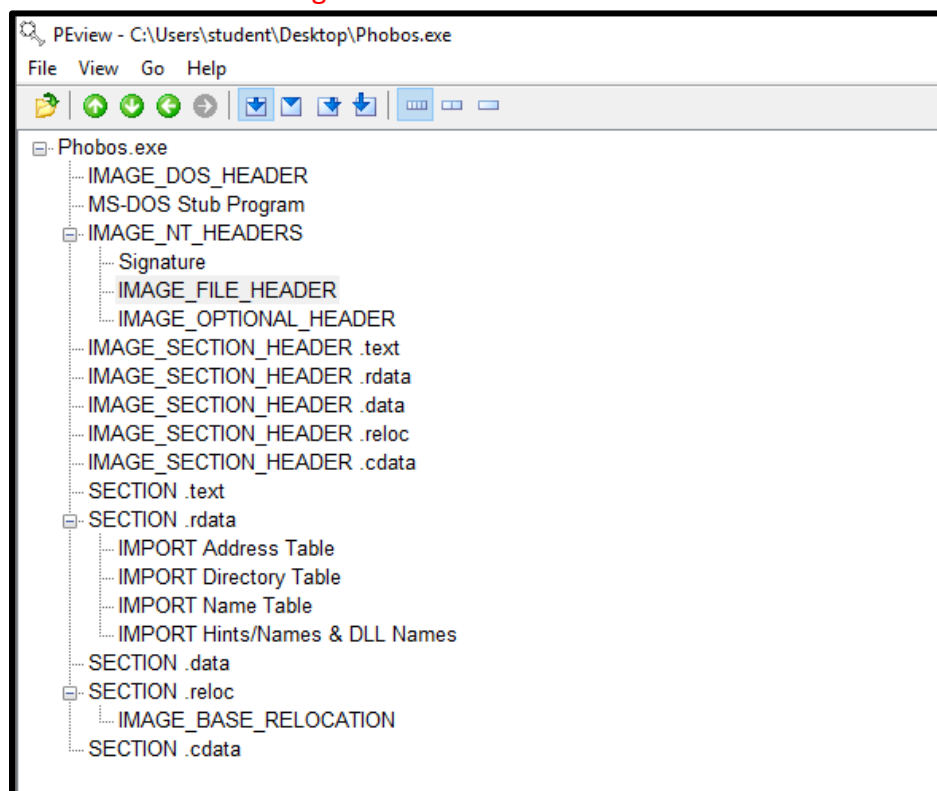


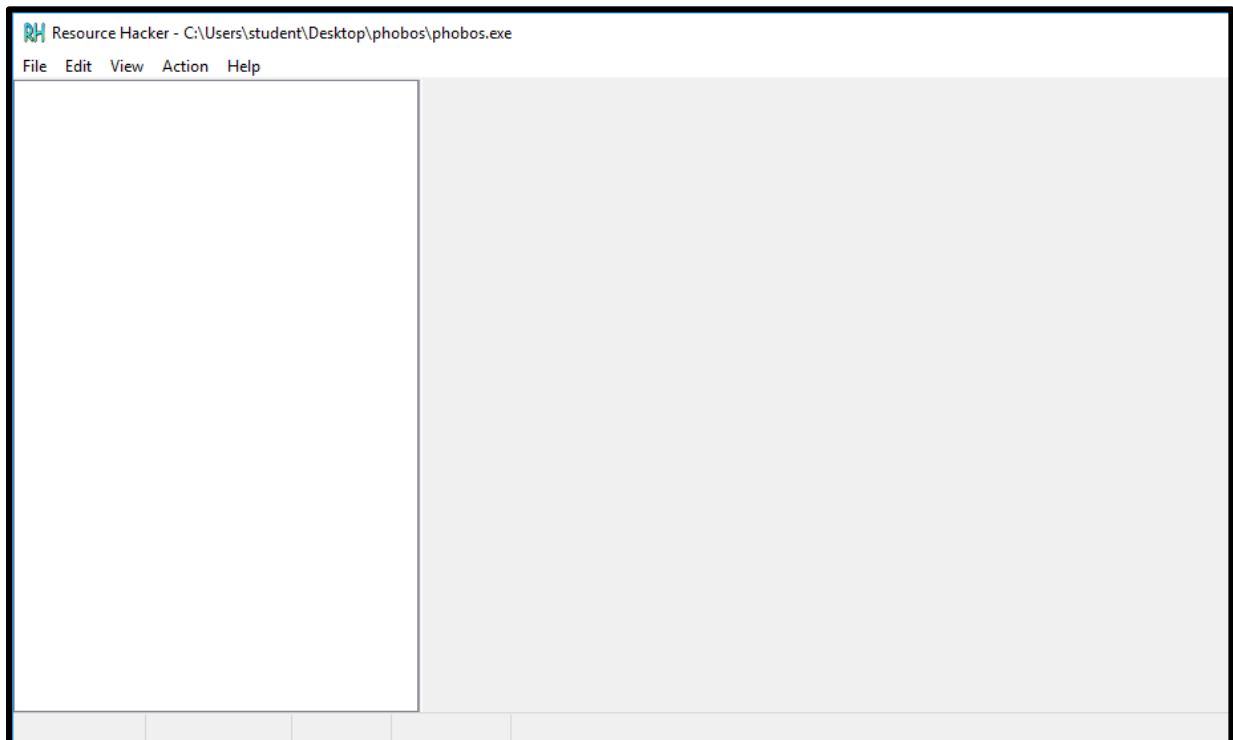
Figure 8: PView File Header

pFile	Data	Description	Value
000000E4	014C	Machine	IMAGE_FILE_MACHINE_I386
000000E6	0005	Number of Sections	
000000E8	5E8350F5	Time Date Stamp	2020/03/31 Tue 14:17:25 UTC
000000EC	00000000	Pointer to Symbol Table	
000000F0	00000000	Number of Symbols	
000000F4	00E0	Size of Optional Header	
000000F6	0102	Characteristics	
		0002	IMAGE_FILE_EXECUTABLE_IMAGE
		0100	IMAGE_FILE_32BIT_MACHINE

Resource Hacker:

A resource hacker is an editor that can be used to decompile and view the different files, executables, and different libraries. As seen in figure 9, when Phobos is run in the resource hacker, nothing is viewable. This could mean a couple of things. It could mean that the malware has anti-debugging on, and therefore when it is put into a decompile, you are unable to view anything. This could also be because it is still packed, which is why we are unable to see anything.

Figure 9: Resource Hacker with Phobos



Basic Dynamic Analysis:


Basic Dynamic Analysis is the process where a malware is examined through the process of running it in a sandboxed environment and tracking its movements. Basic dynamic analysis covers both when the malware is running and the effect it has on the system after it has been fully executed. From this process you will be able to see the functionality of the malware itself such as, files it creates, changes it makes to the system and data it sends. This differs from basic static analysis as there you are only able to see what is visible in the metadata before running the program. This process can be incredibly helpful to get around barriers that you would find in static analysis such as the malware being packed or having anti debugger turned on.

Running The Ransomware:

For the first step of this process, I took a snapshot of my virtual machine to revert it to the state it was before. From this point, the only information that we have about what the malware does is from the virus total, where it claims to be ransomware. We have the DLLs that might point us in the correct direction, but we want to make sure that, regardless of whether we have a backup to revert to. After adding “.exe” to the end of the program, I was able to run the program without any failures. After waiting for around 2 minutes, the message below in figure 10 was launched four times on the desktop. This message states that all the files on the system have been encrypted and states that writing to the email “blockfiles12@tutanota.com” will have them help you retrieve your files. The message also states that if they do not answer in 24 hours, message the other email list to try to gain assistance. Funnily enough, below that, it claims that the price depends on how fast the victim messages them, they expect the victims to be quick, but the ransom gang is unreliable in responding to their email. Under that information, the first box states that the user can get five free files decrypted as long as they didn't have valuable information and were under 4 Mb to show that they are kind. Then in the following box, they link to guides on how to buy bitcoin to pay the ransom. And finally, the red box states not to rename the files or try to decrypt the files using third-party software. Ironically enough, the following line states, “decryption of your files with the help of third parties may cause increased price (they add their fee to our) or you can become a victim of a scam.” Personally, my favorite line I saw well dissecting this malware as it's a complete contradiction to their business model.

Figure 10: Main Ransom Message

encrypted



All your files have been encrypted!

All your files have been encrypted due to a security problem with your PC. If you want to restore them, write us to the e-mail blockfiles12@tutanota.com
Write this ID in the title of your message: **80AF371D-3254**
In case of no answer in 24 hours write us to this e-mail: blockfiles12@cock.li
You have to pay for decryption in Bitcoins. The price depends on how fast you write to us. After payment we will send you the tool that will decrypt all your files.

Free decryption as guarantee

Before paying you can send us up to 5 files for free decryption. The total size of files must be less than 4Mb (non archived), and files should not contain valuable information. (databases, backups, large excel sheets, etc.)

How to obtain Bitcoins

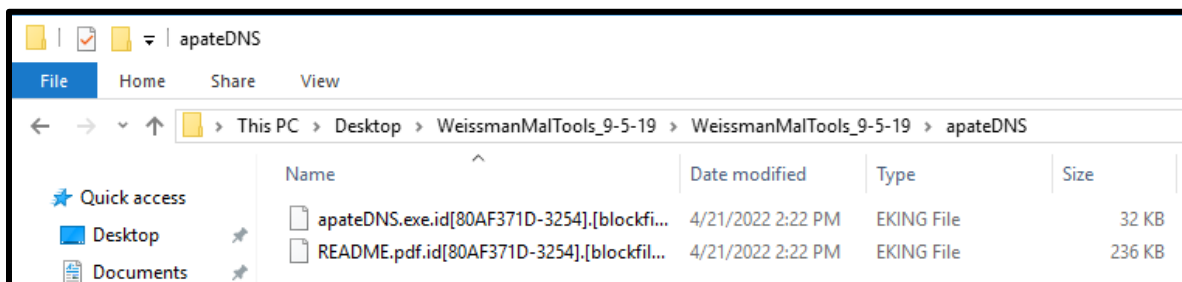
The easiest way to buy bitcoins is LocalBitcoins site. You have to register, click 'Buy bitcoins', and select the seller by payment method and price.
https://localbitcoins.com/buy_bitcoins
Also you can find other places to buy Bitcoins and beginners guide here:
<http://www.coindesk.com/information/how-can-i-buy-bitcoins/>

Attention!

- Do not rename encrypted files.
- Do not try to decrypt your data using third party software, it may cause permanent data loss.
- Decryption of your files with the help of third parties may cause increased price (they add their fee to our) or you can become a victim of a scam.

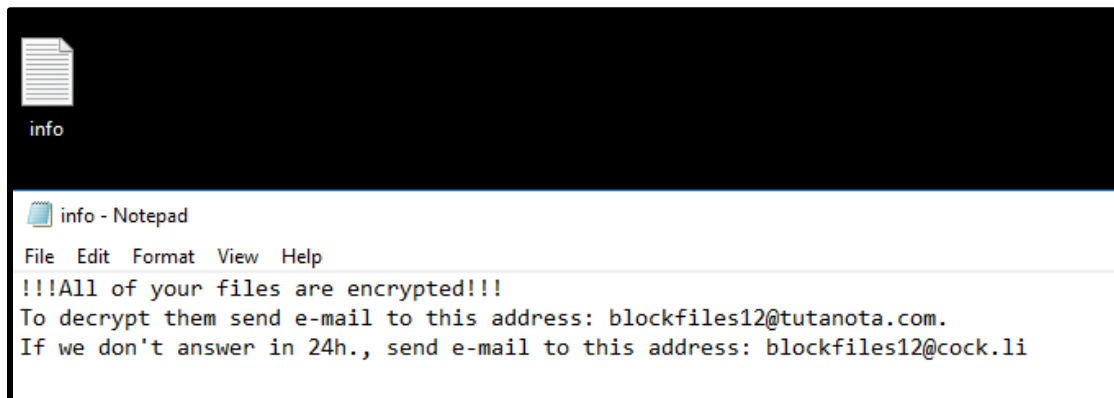
After the messages popped up, I checked and saw if all of my files were encrypted, and sure enough, every file was encrypted. For example, Figure 11 contains two random files from the multitools kit that have been encrypted. I thought an interesting that popped out to me is that the ID mentioned in the ransom messages is included in the name of every encrypted file.

Figure 11: Encrypted Files



Many other interesting, weird things seemed to appear just from what I could see, but what caught my eye the most was two new text documents on my desktop. After opening them, they were both the same thing and contained the following text seen in figure 12. This file contains both the same email addresses as before but seemingly leaves out anything to do with ransom/paying them, and it leaves your ID number.

Figure 12: Secondary Ransom Message



Process Monitor:

Process Monitor better known as procmon is a critical tool for basic Dynamic analysis. This tool tracks registry changes, processes/threads, and the creation and deletion of files on the system. Starting from when you launch the program process monitor saves every single change on the system, making it easy to filter to look for specific changes. This can be helpful when reversing malware as you can see everything that the malware touches, all in one place.

When I first launched Process monitor the First thing that I did was I stopped the capture of events and I cleared out the current list. By the time I had started the program and done that there were already 15 thousand events captured. I turned capture back on and then captured the output until the ransom messages popped back up. The first thing that I noticed was the astronomical amount of events that it had captured. Over a million events had taken place during the almost minute thridy that the process was running as seen in figure 13. To combat this I ended up filtering by the name of the file so I could navigate easier.

Figure 13: Number of Events



One of the first things that I noticed and most important in my opinion was how the ransomware was encrypting all of the files. It would move around the entire system folder by folder until it came across a file it could encrypt that wasn't a folder. When it entered a new folder it would use "QueryBasicInformationFile" where it would then check to see if the file could be encrypted. If it could it used "CreateFile" to do so. If there was nothing at the end of a folder path it would use "CloseFile" and back out. This is what created all of the events as this happened for every single folder and file in the system. Some examples of this can be seen below in figure 14.

Figure 14: Locking the Files

Time ...	Process Name	PID	Operation	Path
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Pictures\Saved Pictures\desktop.ini
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Pictures\Saved Pictures\desktop.ini.id[80AF371D-3254].[blockfiles12@tutanota.com].eking
2:45:3...	Phobos.exe	6964	CreateFile	C:\Users\student\Pictures\Saved Pictures\desktop.ini
2:45:3...	Phobos.exe	6964	QueryAttribute T...	C:\Users\student\Pictures\Saved Pictures\desktop.ini
2:45:3...	Phobos.exe	6964	SetDispositionI...	C:\Users\student\Pictures\Saved Pictures\desktop.ini
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Pictures\Saved Pictures\desktop.ini
2:45:3...	Phobos.exe	6964	CreateFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	QueryStandardl...	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CreateFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	QueryBasicInfor...	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CreateFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	QueryBasicInfor...	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CreateFile	C:\Users\student\Saved Games\desktop.ini.id[80AF371D-3254].[blockfiles12@tutanota.com].eking
2:45:3...	Phobos.exe	6964	CreateFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	QueryStandardl...	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CreateFile	C:\Users\student\Saved Games\desktop.ini.id[80AF371D-3254].[blockfiles12@tutanota.com].eking
2:45:3...	Phobos.exe	6964	ReadFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	WriteFile	C:\Users\student\Saved Games\desktop.ini.id[80AF371D-3254].[blockfiles12@tutanota.com].eking
2:45:3...	Phobos.exe	6964	WriteFile	C:\Users\student\Saved Games\desktop.ini.id[80AF371D-3254].[blockfiles12@tutanota.com].eking
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Saved Games\desktop.ini
2:45:3...	Phobos.exe	6964	CloseFile	C:\Users\student\Saved Games\desktop.ini.id[80AF371D-3254].[blockfiles12@tutanota.com].eking

Another interesting thing found within the events was the constant use of the file "mshta.exe." It can be seen used a few times in figure 15 below. This was not the only section it was located that peaked my interest. After looking it up, I found out that it stood for Microsoft HTML application host. This led me to believe that the ransomware was using this exe to attempt to reach something on the internet.

Figure 15: mshta.exe

Time ...	Process Name	PID	Operation	Path	Result
2:45:3...	Phobos.exe	6964	RegQueryValue	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\mshta.exe\MitigationOptions	SUCCESS
2:45:3...	Phobos.exe	6964	QueryNameInfo...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	Process Create	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders	SUCCESS
2:45:3...	Phobos.exe	6964	RegSetInfoKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders	SUCCESS
2:45:3...	Phobos.exe	6964	RegQueryValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Cache	SUCCESS
2:45:3...	Phobos.exe	6964	RegCloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders	SUCCESS
2:45:3...	Phobos.exe	6964	RegQueryKey	HKCU\Software\Microsoft\Windows NT\CurrentVersion	SUCCESS
2:45:3...	Phobos.exe	6964	RegOpenKey	HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers	NAME NOT FOL
2:45:3...	Phobos.exe	6964	QuerySecurityFile	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	CreateFile	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	QueryBasicInfor...	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	CloseFile	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	CreateFile	C:\Windows\AppPatch\apppatch64\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	QueryBasicInfor...	C:\Windows\AppPatch\apppatch64\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	CloseFile	C:\Windows\AppPatch\apppatch64\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	QueryBasicInfor...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	QueryBasicInfor...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	QuerySecurityFile	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	QueryNameInfo...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	CreateFile	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	QueryStandardI...	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	QueryStandardI...	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	CreateFileMapp...	C:\Windows\AppPatch\sysmain.sdb	FILE LOCKED W
2:45:3...	Phobos.exe	6964	QueryStandardI...	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	CreateFileMapp...	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	QueryStandardI...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers	NAME NOT FOL
2:45:3...	Phobos.exe	6964	RegOpenKey	HKCU\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Layers	NAME NOT FOL
2:45:3...	Phobos.exe	6964	RegOpenKey	HKLM\Software\Microsoft\Windows NT\CurrentVersion\AppCompatFlags\Custom\mshta.exe	NAME NOT FOL
2:45:3...	Phobos.exe	6964	QueryStandardI...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	CreateFileMapp...	C:\Windows\SysWOW64\mshta.exe	FILE LOCKED W
2:45:3...	Phobos.exe	6964	QueryStandardI...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	CreateFileMapp...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	ReadFile	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	ReadFile	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	CloseFile	C:\Windows\AppPatch\sysmain.sdb	SUCCESS
2:45:3...	Phobos.exe	6964	QueryBasicInfor...	C:\Windows\SysWOW64\mshta.exe	SUCCESS
2:45:3...	Phobos.exe	6964	RegOpenKey	HKLM\Software\WOW6432Node\Microsoft\Windows\CurrentVersion\SideBySide	NAME NOT FOL
2:45:3...	Phobos.exe	6964	RegOpenKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders	SUCCESS
2:45:3...	Phobos.exe	6964	RegSetInfoKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders	SUCCESS
2:45:3...	Phobos.exe	6964	RegQueryValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Cache	SUCCESS
2:45:3...	Phobos.exe	6964	RegCloseKey	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders	SUCCESS
2:45:3...	Phobos.exe	6964	QuerySecurityFile	C:\Windows\SysWOW64\mshta.exe	SUCCESS

That was until I dug a little further and saw what was pictured in figure 16. Where mshta.exe creates a file that happens to be the new location of the desktop after the ransomware is run. The first ransomware message popped up again after going to the location and running this file. So the ransomware could very well be using this exe to connect to the internet, but from this, I can conclude it's using HTML to build its ransom messages and using internet explorer to do it.

Figure 16: info.hta

2:45:3...	mshta.exe	1568	RegCloseKey	HKLM\SOFTWARE\W
2:45:3...	mshta.exe	1568	CreateFile	C:\info.hta
2:45:3...	mshta.exe	1568	QueryBasicInformationFile	C:\info.hta
2:45:3...	mshta.exe	1568	CloseFile	C:\info.hta
2:45:3...	mshta.exe	1568	RegOpenKey	HKCU\Software\W

Process Explorer:

Process Explorer is almost like a better version of the windows task manager. This tool allows the user to get a more in-depth view of the process that is running on the system. Along with the name of everything currently running, the user can see the file paths, the creator or source, a short description, and you can even tell if the program is malicious or not. One of the most significant advantages of using this software is that when set into DLL mode, the user can see a memory map of any DLL that a process has touched and what functions it used within those DLLs.

When Phobos was run with process explorer open, there was not much noticeable change to the main menu. The interesting things started to be seen when you dug deeper into the running programs. One thing that happened that I didn't consider at first is how the ransomware changes the permissions of my current user. When I looked at the individual programs running, I noticed this. First, as seen in figure 17, I cannot see the job name, most likely because of my lack of permission. This happened again in figure 18 with a random exe file where I could not load the strings because I didn't have the correct access to view them.

Figure 17: Access Denied Process Explorer

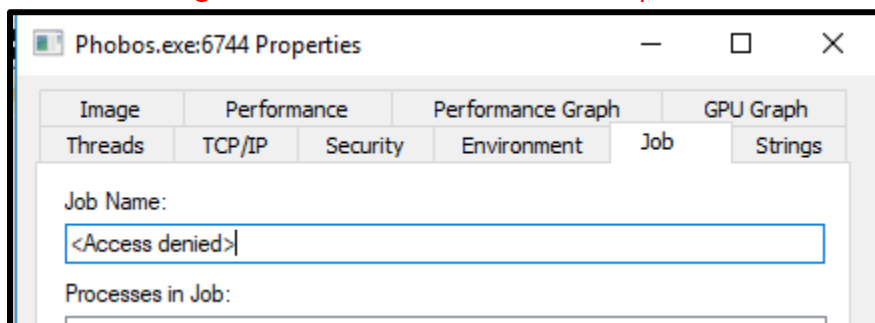
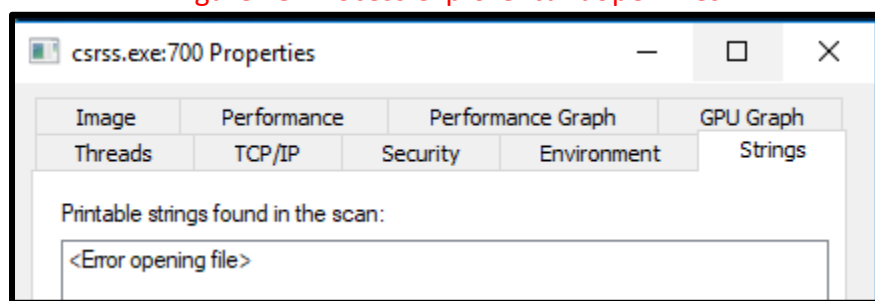


Figure 18: Process explorer cant open files



One last thing is that the suspicions I had about mshta being used to craft the ransom note proved correct when looking at it through process explorer. After letting the ransomware run to completion and expanding the exe, these are all listed as threads underneath. This would prove my thoughts before correct.

Figure 19: mshta.exe again

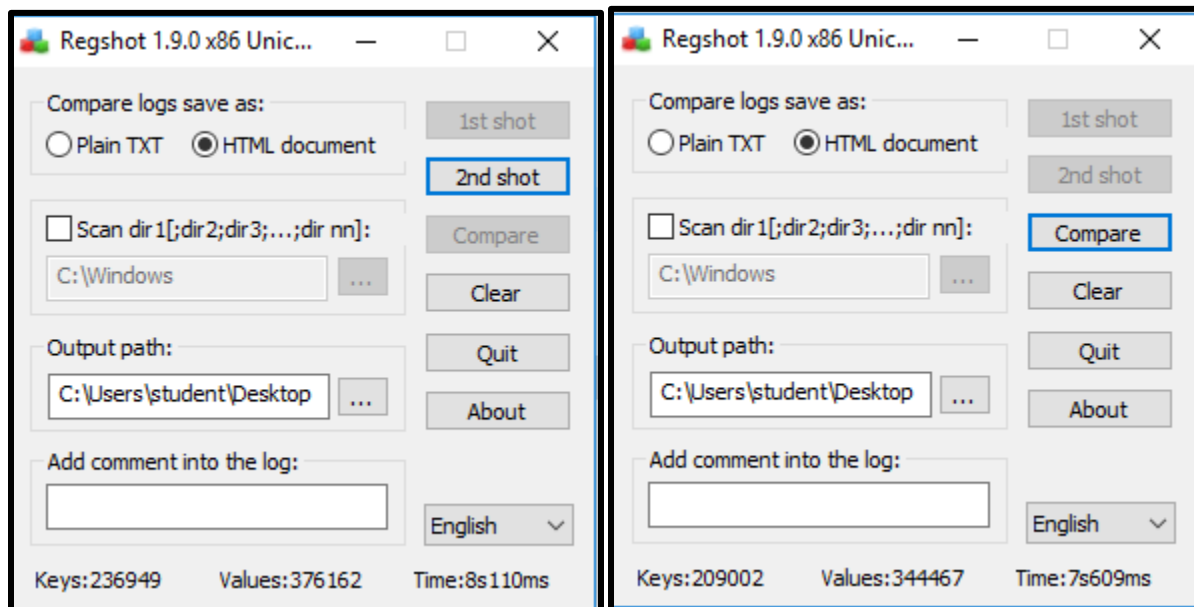
Phobos.exe	1.10	3,484 K	13,580 K	488
mshta.exe		11,348 K	39,676 K	2852
mshta.exe		11,312 K	39,684 K	1568
mshta.exe		11,364 K	40,136 K	6408

RegShot:

Regshot is a program that allows the user to take an inventory of their entire registry, run a program, and then retake inventory of their registry. From there, regshot then compares the two snapshots of registers and puts all of the changes into a convenient HTML document to read. This can be helpful when doing basic dynamic analysis as you can take a snapshot of your registry before running the malware. Then after it has been successfully running, you can retake the snapshot and see the exact values that the malware went and changed in the system.

Before rerunning the ransomware, I pressed the first shot button to get an inventory of the registry keys before they changed. This ended up with counting around 240000 keys. After the ransomware was run, it can be seen that the total amount of keys dropped drastically, to around two hundred thousand. This means that all of those registry changes from the process monitor must have been the deletion of all of these keys, as we can see here.

Figure 20: First and Second Shots



When the two inventories are then compared, we can see right at the top the total number of deleted keys. It seems that the ransomware went through and mainly deleted many of the extra drivers that were in the registry. Strangely enough, most of the drivers that I recognized were for printers and scanners, but I don't completely understand why the ransomware would be looking to delete those. In Figure 21, we can see just a sample of the number of keys that were actually deleted.

Figure 21: Keys deleted

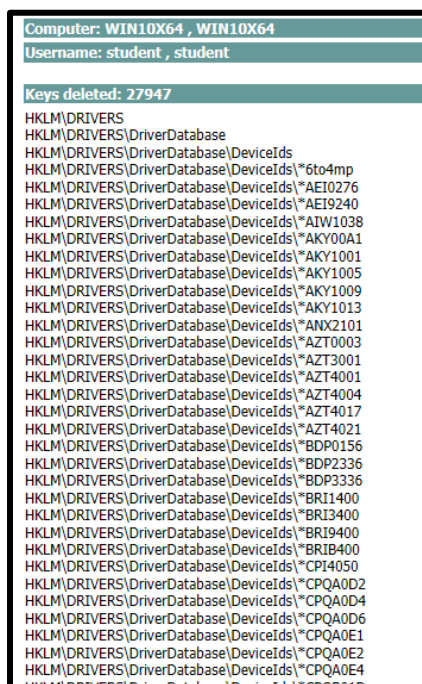
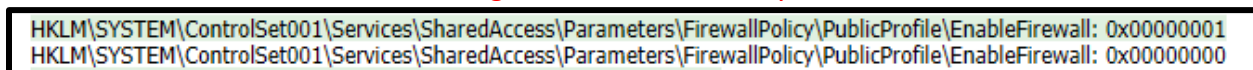


Figure 21 came from the modified keys section of the document. This shows that the ransomware went to the firewall and disabled it. This could be for the ability to connect to outside networks for transferring data.

Figure 22: Firewall Policy



Finally, arguably one of the most important finds about this ransomware was found from using regshot in figure 23. Below we can see that this is where the ransomware is able to detect that it is running on a VM. This is important for a lot of different reasons. First, if the malware knows that it's running on a VM, there is a chance it could change the way that it interacts with its environment, making the analyst not able to get the complete picture of what it does. The second reason that this is important is that this means that this ransomware is very sophisticated. Not many malware can detect this, and because Phobos can mean that the advanced analysis is most likely going to be a lot harder as it doesn't seem this malware wants

to be deconstructed. This fact, combined with the information about the anti-debug found in the basic static analysis combined, makes this ransomware extremely complex.

Figure 23: Hypervisor Detected

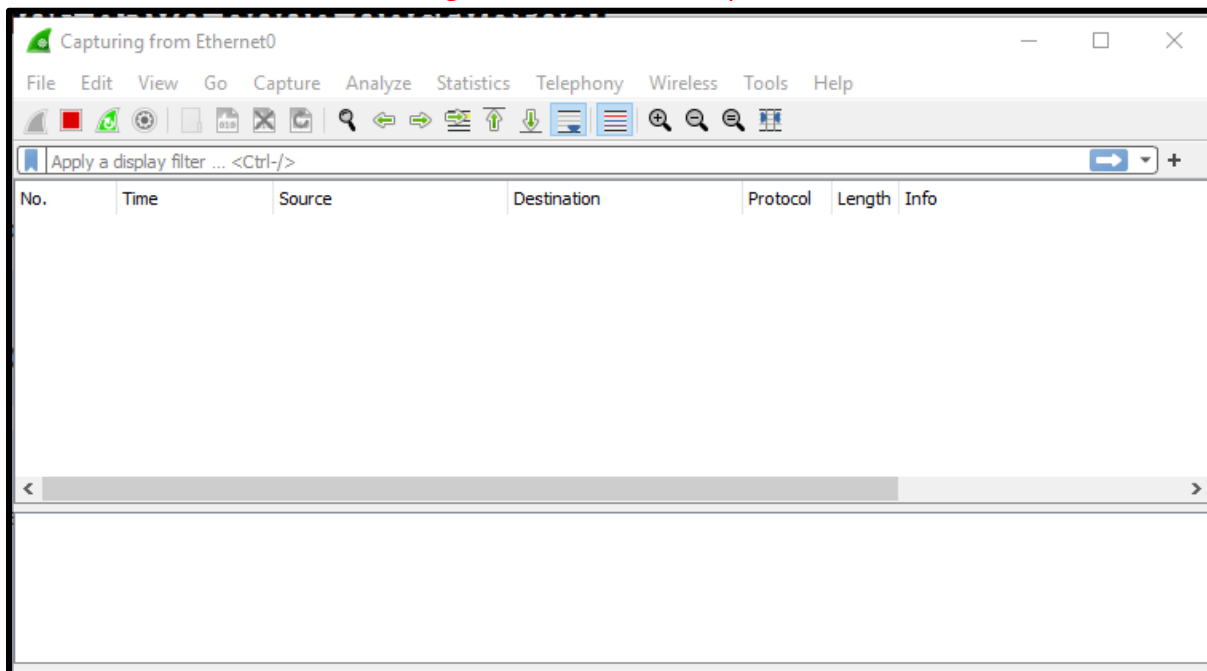
```
"vendorId="0x1414",deviceID="0x8c",subSysID="0x0",revision="0x0",version="10.0.15063.0"hypervisor="Hypervisor detected (No SLAT)""  
"vendorId="0x1414",deviceID="0x8c",subSysID="0x0",revision="0x0",version="10.0.15063.447"hypervisor="Hypervisor detected (No SLAT)""
```

Wireshark:

Wireshark is another networking tool that can be useful in seeing what exactly a program is trying to reach. This would be important when malware is trying to communicate with something outside of the network, whether this is a website or possibly another computer. Wireshark has many useful features for this, such as filtering through packets for a specific type. Wireshark also can look into the data of the packets. The packets will have IP addresses, mac addresses, etc., that can be seen in plaintext, which will give us a better idea of what the malware is doing. If the malware is created poorly, you might be able to see plaintext data that it is trying to pull in or push out of the system so you can know what it is taking or bringing in.

Although we know that the ransomware is using HTTP at some points, when it is run well using a Wireshark trace, there are no packets. This could also be a factor in the information found before knowing it's in a virtual machine because it might not waste the resources of trying to connect to the network if it knows that it can't, which would be why we can't see anything.

Figure 24: Wireshark capture



Advanced Static Analysis:

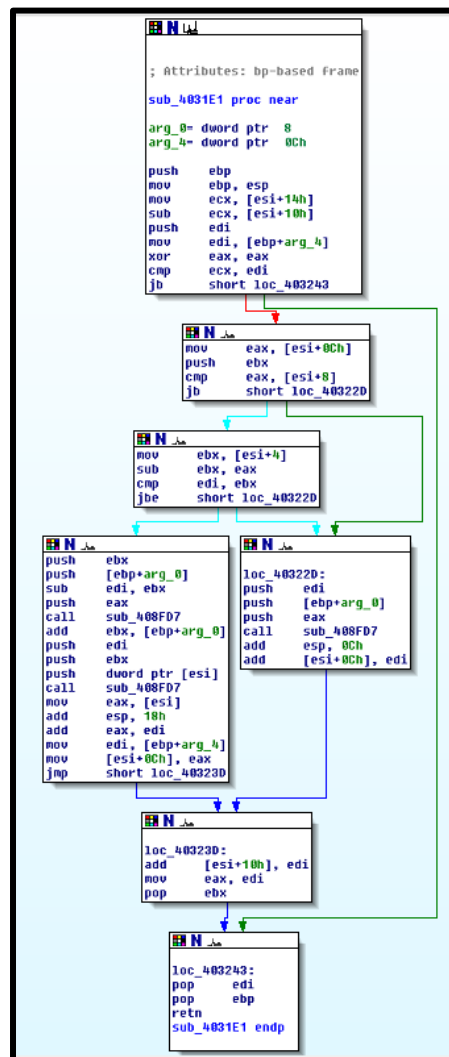
Advanced static analysis should be the third step in the malware reversing process. This process entails putting the malware into a disassembler. This will then allow you to look at the individual instructions that the program executes well it's running. This is one of the best ways to get the most intimate details of what the program is doing as you are looking at the code at an assembly level.

IDA Pro:

Ida pro is one of the main disassemblers used for malware analysis. Ida pro allows the user to see the individual assembly code, but it also allows the user to see a flow chart graph. In this graph, the user can see functions and what they do/ how they interact with each other. The only issue is it's very hard to read as you must know how to read assembly. And there is a chance that the disassembled code is not accurate as it's just an analysis of how the computer thinks the source code reads.

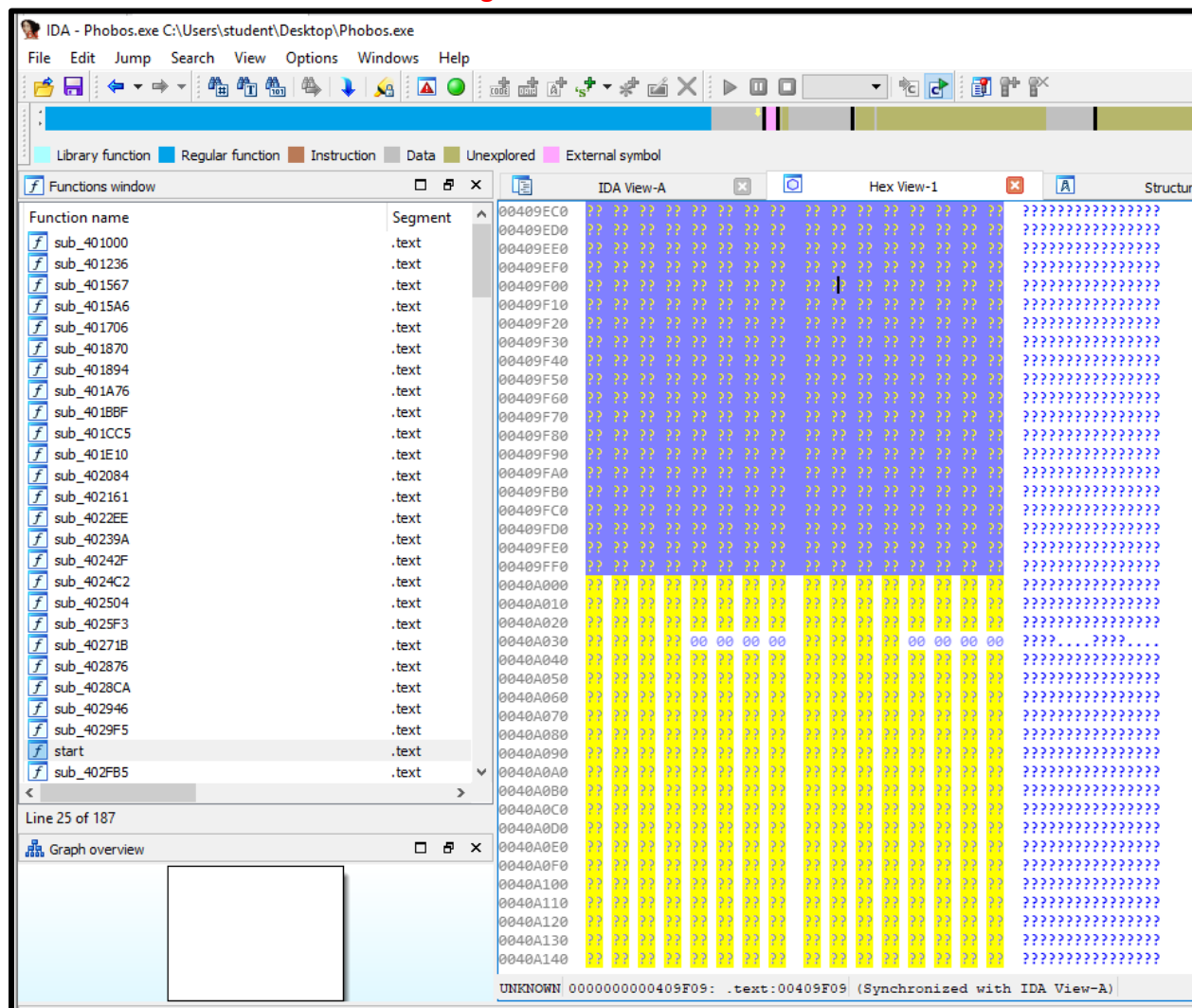
When I put Phobos into Ida pro, everything seemed to be expected as I was able to read the first flow chart statement fine by pressing space. That is seen below in figure 25. Because it is making calls to other parts of the code that I cannot see, I cannot be entirely sure what this flow chart is depicting.

Figure 25: Ida Flow Chart



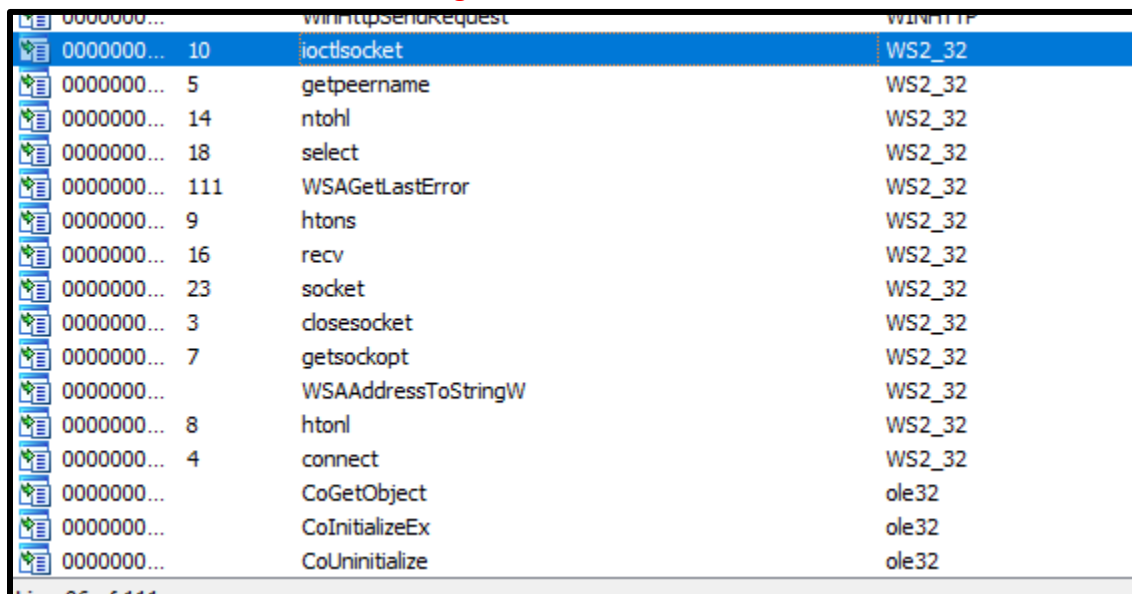
And the main reason why I am not able to see any of the other sections of code is because of what is pictured in figure 26. When looking at the Hex, it has turned into question marks. Most likely because of the anti-debugging, when the source code was decompiled, it must have been obscured so that the primary function calls within the program were not exposed, so the user couldn't see exactly what it does. Many things could be hidden within this obscured code, such as hidden functions, passwords, or affiliated IP addresses/websites. But without knowing how to avoid these, I cannot proceed.

Figure 26: XORed data



Although even with that defeat, I was able to find out more information about the source code itself. We can accurately guess from the function names below that the code ransomware was written in C# as these functions are directly from the said language.

Figure 27: C# Evidence



00000000...	10	WinHttpSendRequest	WS2_32
00000000...	5	ioctlsocket	WS2_32
00000000...	14	getpeername	WS2_32
00000000...	18	ntohl	WS2_32
00000000...	111	select	WS2_32
00000000...	9	WSAGetLastError	WS2_32
00000000...	16	htons	WS2_32
00000000...	23	recv	WS2_32
00000000...	3	socket	WS2_32
00000000...	7	closesocket	WS2_32
00000000...	8	getsockopt	WS2_32
00000000...	4	WSAAddressToStringW	WS2_32
00000000...	8	htonl	WS2_32
00000000...	4	connect	WS2_32
00000000...		CoGetObject	ole32
00000000...		CoInitializeEx	ole32
00000000...		CoUninitialize	ole32

Advanced Dynamic Analysis:

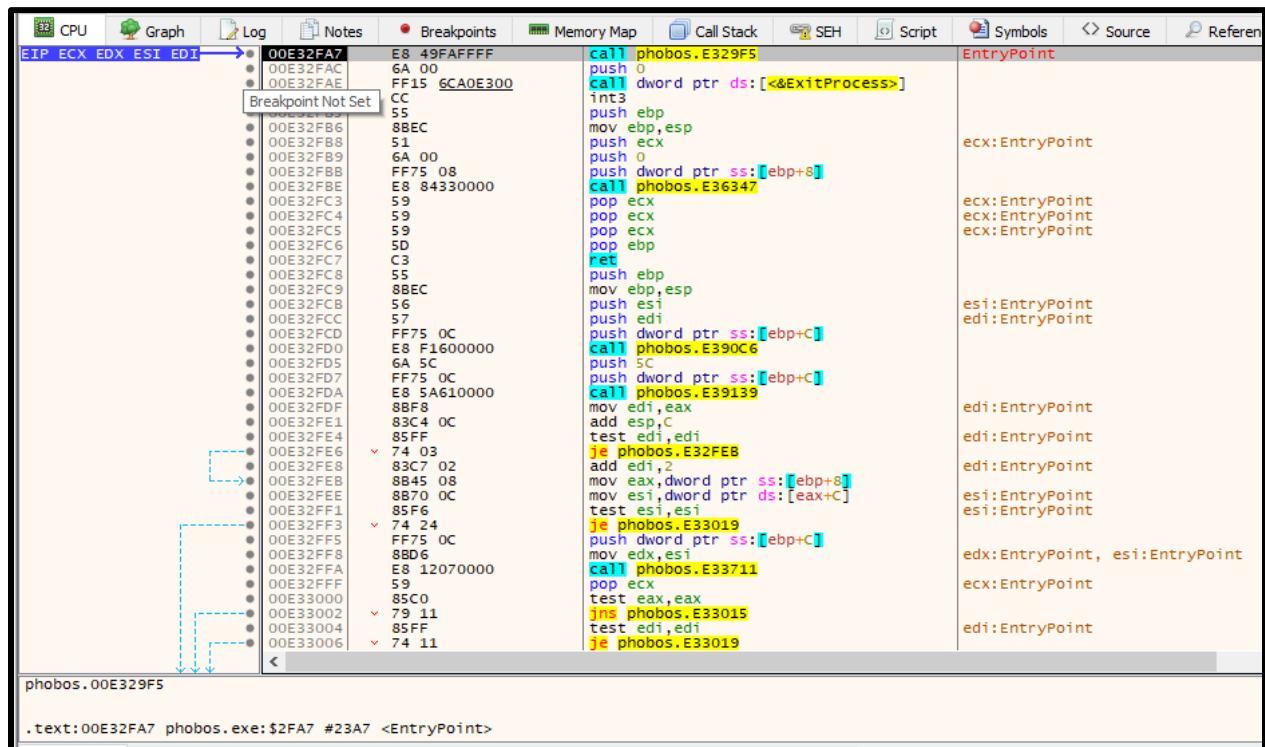
Advanced Dynamic Analysis, like advanced static, takes a look at the assembly code of the program. In this case, the code is put into a debugger, allowing the user to place breakpoints at specific parts of the program and manipulate assembly code to produce different outputs. This could give you a better idea of how the program was built. One example of this is if there is something in the password-protected malware, the user can manipulate the code to either find the password or change the password to be their own within the assembly. Another reason would be to slow the program down. By placing breakpoints, you can stop the program from running to evaluate the changes it is making as its making them.

X32dbg:

Finally, the last tool that we will be using is x32dbg. In this tool, the user can step through each program's assembly code line. This allows them to place breakpoints and discover what is happening within the malware's code in real-time.

When I put Phobos into this tool, everything looked normal at first, just like it did when put into Ida. I could press the start button and get to the call where the function would be started, as seen in figure 28. Here at the entry point, after pressing the step into the button, the program would start, and we would be allowed to individually step through each line to see the changes and what is happening. From the code below, everything looks normal.

Figure 28: x32dbg starting point



That is until we press the button, and we are brought all the way down to the place where the call is located. As seen in figure 29, there are many “int 3,” which seems like a bad sign as no standard assembly code would have anything like those multiple times in a row. After pressing the step into button one more time, we are met with the message seen at the bottom of the screenshot “First chance exception on 7628B802 (000006Ba, RPC+S_SERVER_UNAVAILABLE)!” After doing some research, this error is intentionally put into code to stop it from running, usually by someone debugging the software to allow it to catch. After pressing the step into button multiple more times with the same error occurring, I can safely say that the error is most likely because of the anti-debugging that we discovered earlier. I feel that it was intentionally put in there to stop anyone from being able to debug the program easily before they discovered something that would lead to the people in charge of the ransomware getting caught. Because I am not at a level of being able to undo this either, I am again stuck.

Figure 29: x32dbg stopping point

Phobos.exe - PID: 18C0 - Module: kernelbase.dll - Thread: 174 - x32dbg

File View Debug Trace Plugins Favourites Options Help Aug 23 2019

CPU Graph Log Notes Breakpoints Memory Map Call Stack SEH Script

EIP → 762887E0 77 38 ja kernelbase.7628881A
 762887E2 894424 10 mov dword ptr ss:[esp+10],eax
 762887E6 C1E0 02 sh1 eax,2
 762887E9 50 push eax
 762887EA 51 push ecx
 762887EB 8D4424 1C lea eax,dword ptr ss:[esp+1C]
 762887EF 50 push eax
 762887F0 E8 CF9C0100 call <JMP.&memcpy>
 762887F5 83C4 0C add esp,C
 762887F8 8D0424 lea eax,dword ptr ss:[esp]
 762887FB 50 push eax
 762887FC FF15 90533376 call dword ptr ds:[<&RtlRaiseException>
 76288802 884C24 54 mov ecx,dword ptr ss:[esp+54]
 76288806 33CC xor ecx,esp
 76288808 E8 336A0100 call kernelbase.762A2240
 7628880D 8BE5 mov esp,ebp
 7628880F 5D pop ebp
 76288810 C2 1000 ret 10
 76288813 836424 10 00 and dword ptr ss:[esp+10],0
 76288818 EB DE jmp kernelbase.762887F8
 7628881A 6A 0F push F
 7628881C 58 pop eax
 7628881D EB C3 jmp kernelbase.762887E2
 7628881F CC int3
 76288820 CC int3
 76288821 CC int3
 76288822 CC int3
 76288823 CC int3
 76288824 CC int3
 76288825 CC int3
 76288826 CC int3
 76288827 CC int3
 76288828 CC int3
 76288829 CC int3
 7628882A CC int3
 7628882B CC int3
 7628882C CC int3
 7628882D CC int3
 7628882E CC int3
 7628882F CC int3
 76288830 6A 2C push 2C

ecx=0
 dword ptr [esp+54]=[0254F36C]=C888513E
 .text:76288802 kernelbase.dll:\$EB802 #EAC02

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 [x=] Locals Struct

Address	Hex	ASCII
770D1000	1C 00 1E 00 94 82 0D 77 28 00 2A 00 68 82 0D 77w(*.h..w
770D1010	34 00 36 00 30 82 0D 77 1E 00 20 00 10 82 0D 77	4.6.0..w....w
770D1020	1A 00 1C 00 F4 81 0D 77 18 00 1A 00 08 81 0D 77	...0..w....0..w
770D1030	20 00 22 00 84 81 0D 77 30 00 32 00 80 81 0D 77	..".w0.2...w
770D1040	2C 00 2E 00 50 81 0D 77 20 00 22 00 2C 81 0D 77	...P..w"..w
770D1050	18 00 1A 00 10 81 0D 77 10 00 12 00 FC 80 0D 77w....ü..w
770D1060	0E 00 10 00 98 83 0D 77 00 00 02 00 88 5F 0D 77w...._..w
770D1070	10 00 12 00 84 82 0D 77 0C 00 0E 00 88 83 0D 77w....w
770D1080	06 00 08 00 68 83 0D 77 06 00 08 00 78 83 0D 77	...h..w....x..w
770D1090	06 00 08 00 80 83 0D 77 06 00 08 00 70 83 0D 77w....p..w
770D10A0	16 00 18 00 80 7F 0D 77 1C 00 1E 00 7C C0 0D 77	...w.... A..w
770D10B0	04 00 06 00 D4 7F 0D 77 18 00 00 00 00 00 00 00	...0..w.....
770D10C0	6C 17 0D 77 40 00 00 00 00 00 00 00 00 00 00 00	l..we.....
770D10D0	40 88 10 77 10 83 10 77 18 00 1A 00 60 C0 0D 77	@..w....w...A..w
770D10E0	14 00 16 00 48 C0 0D 77 08 00 0A 00 C8 7F 0D 77	...HÄ.w....É..w
770D10F0	08 00 0A 00 A0 80 0D 77 02 00 04 00 50 83 0D 77	...W....w...P..w
770D1100	00 CF 10 77 10 88 10 77 80 48 1C 77 60 48 1C 77	↑..w...w..T..w'K..w

Command: First chance exception on 76288802 (000006BA, RPC_S_SERVER_UNAVAILABLE)!

Paused

Conclusory Analysis:

Although I couldn't get past the anti-debugging or the anti-VM measures that the ransomware had in place, I think that even with just the information I presented, I can safely say that this is a very high-level Ransomware attack. A few of the conclusions I was able to come to regarding this malware was...

1. It is a complex Ransomware
2. It has Anti-Debugging and Anti-Virtual Machine tactics
3. It is written In C#
4. The Ransome note is written using HTML specifically internet Explorer

Even with just my semester of using tools like this, I am proud that I could get as far into reversing the malware as I did, especially with this being coded so toughly. Backed with the information on all the tools that I used to dissect this ransomware, I think that with a little more knowledge, I would be able to get into the malware fully, and this project inspired me to keep learning to do that so at some point, I can with ease.

Potential dangers:

There are quite a few potential dangers that could come about from this specific ransomware. If this were to get onto a system of great importance, such as a hospital or government computer system, it could cause a great deal of harm by completely halting work. This is because most jobs today revolve around files on a computer, and without access, those jobs cant be completed. This could lead to the loss of billions of dollars or even lives with the hospital example. Luckily Phobos wasn't used on any reported hospitals. However, it still impacted many smaller businesses that couldn't handle losing all of their files or paying the considerable amounts asked of them when paying the ransom was their last option. If this happened to get onto a singular system, the worst-case scenario is that all of your files become unusable, but that would still be something to avoid.

Mitigation:

Most of the basic computer etiquette works when trying to avoid a ransomware attack such as this. These would be things like not clicking on random links or downloading things from places you don't know. One thing that could be useful in preventing something like this would be to make a backup often. If you make a habit of backing up your systems at least once a week or so, the likelihood that a ransomware attack would cause a significant issue would go down tremendously. If you have multiple backups of all the files, if worst cases scenario happens, you can revert to the backup. Some ideas for a more business-minded way to mitigate ransomware attacks is, of course, to teach employees basic internet safety. By teaching and reminding them about the dangers of ransomware, they would be less likely to do something that would let ransomware in accidentally. Blocking all unneeded extensions and executables of employee computers could be an excellent way to prevent anything from spreading unnesscarly. One last tip that I think is most important when thinking about these attacks is to make sure that you protect your computer privilege. Many ransomware, especially Phobos, pray on systems with weak privilege in their users, and it can exploit that to gain access to your entire system. If you can keep that secure, the likelihood that a ransomware attack such as this would be damaging goes down entirely.

Citations:

“Overview of Phobos Ransomware.” *HHS Cybersecurity Program*, HHS Office of Information Security, 7 July 2021, www.hhs.gov/sites/default/files/overview-phobos-ransomware.pdf.

“Phobos Ransomware (Analysis and Recovery Options).” *Proven Data*, 13 Mar. 2021, www.provendatarecovery.com/phobos-ransomware-recovery.

Ingalls, Sam. “How to Prevent Ransomware Attacks: 20 Best Practices for 2022.” *eSecurityPlanet*, 20 Dec. 2021, www.esecurityplanet.com/threats/ransomware-protection.

Strawbridge, Geraldine. “The Dangers Of Ransomware.” *MetaCompliance*, 17 Feb. 2020, www.metacompliance.com/blog/the-dangers-of-ransomware.

“First-Chance Exception at in : 0x000006BA: The RPC Server Is Unavailable.” *Stack Overflow*, 15 Oct. 2008, stackoverflow.com/questions/204178/first-chance-exception-at-addr-in-myapp-0x000006ba-the-rpc-server-is-unava#204436.

“What Is Mshta.Exe?” *File.Net*, www.file.net/process/mshta.exe.html. Accessed 30 Apr. 2022.

Stevewhims. “About WinHTTP - Win32 Apps.” *Microsoft Docs*, 7 Jan. 2021, docs.microsoft.com/en-us/windows/win32/winhttp/about-winhttp.