Course Project: Milestone 3

Cesar Munguia

04/02/2024

—

IS-4543-001

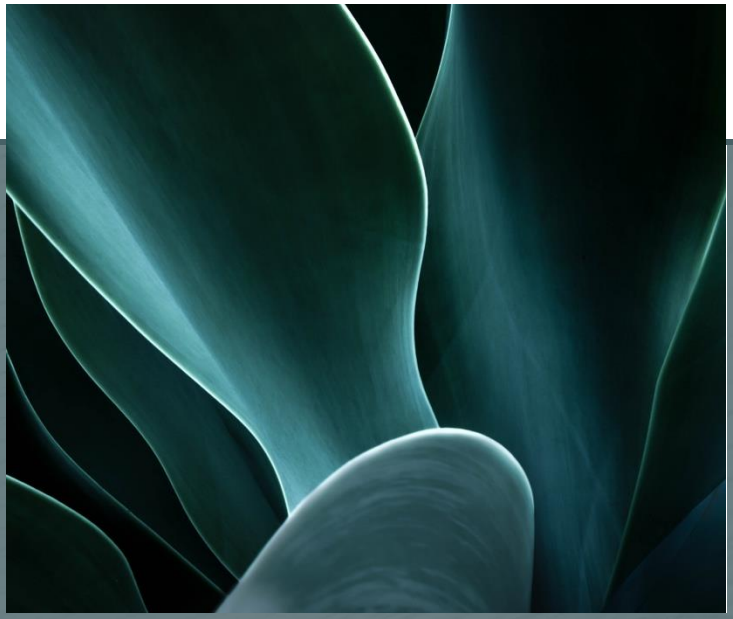
—

Jose Mireles

Objectives

Each milestone submission will be a write-up that includes:

- What did you do?
- What did you learn?
- Documentation of your work
  - Screenshot
  - Picture
  - Video
  - Interpretive Dance



Project Proposal

The project involves in-depth analysis of a virtual machine infected with malware, focusing on understanding the malware's behavior, persistence mechanisms, and potential impact on the system. Through forensic techniques and dynamic analysis, I will dissect the malicious code, identify evasion tactics, and extract indicators of compromise. The project aims to enhance cybersecurity skills by providing hands-on experience in malware analysis and incident response, crucial for defending against and mitigating the effects of sophisticated cyber threats.

- Milestone 3: Conduct a static analysis of the chosen malware file using several tools

In Milestone 3 I will be focusing in conducting a static analysis of the malicious file to gain a better understanding of the overall behavior. With this I will have an idea of what the program does, what imports it might use, what type of capabilities it might have, and what connections it might make to name a few characteristics. In this milestone I decided to not use the original malware that was intended for inspection due to VM-aware capabilities. For that reason, I downloaded several malware databases to play around with them and confirm that they executed accordingly and did not have VM-aware capabilities.

Before moving on, I'd like to set the stage for the activities I'm about to do. Let's say that in a hypothetical realistic world, an employee from Company XYZ receives a spear-phishing email targeted originally for that individual only. Due to the person's ignorance and unawareness about the topic, he believed everything what the email said. In the email, this other individual claimed to be an upper-level officer requesting the employee to download a program/utility needed for one of his upcoming tasks. Little did the person know, that a malicious program was embedded in the .exe file that also executes when double-clicking the program icon. Later, the same employee reported to the

IT department that his computer had been acting strange and asked specialized personnel to take a look at his computer. Here, is where I come in with the analysis.

Summary of Activities

As soon as I downloaded the malware databases, I took a snapshot of my virtual machine in case anything went wrong afterwards (refer to screenshot 1).

Ignoring the previous step and coming back to the hypothetical realistic scenario of the employee of Company XYZ, I asked the employee to state the last things he did while using the computer. He said that the computer was running fine until he downloaded the double-clicked the suspicious program named "Core1Installer.exe" from the email. Then, I made the assumption that this program was the one responsible of all abnormalities. I decided to take a static analysis on this .exe file and gain valuable and significant data.

My first step was to conduct a similarity test, meaning I would compare it to other common and well-known malware out there. From the database of malware, I made a .txt file with several hashes of malware. I then used tlsh.exe in the command prompt to give me a measure of how similar this program hash was compared to other hashes (refer to screenshot 2). The command used was **tlsh -c Core1Installer.exe -l malware_digest.txt**. Then, I used another command-prompt program called capa.exe. I first placed the capa binary in the same directory as the malware file and then ran the command **capa Core1Installer.exe > malware_capa.txt** (refer to screenshot 3 and 4). My next step was to use another command-prompt program called bstrings.exe to see if I was able to find some significant plaintext strings in the executable. First, I used command **bstrings.exe -p**, which showed me the type of strings it can look for in the file (refer to screenshot 5). I initially used the following options: ipv4, win_path, reg_path, and b64, but only got some data back from the first two (refer to screenshots 6 & 7). Unfortunately, I was not able to find wallets of any sort (bitcoin, bytecoin, etc.), URLs, MAC addresses, or emails. Then, I opened up PEid to scan the file for extra information

(refer to screenshot 9). I also used PEstudio to give even more information. While at the GUI, I went to the “Indicators” tab, and analyzed only indicators with a severity of level 1 and 2 (refer to screenshot 10). Finally, I used the Ghidra program to help me try disassemble the code. Inside the program I took a peek to the decompile code on the far-right side (refer to screenshot 13). The program itself was too large to be fit in one screenshot. Then I looked at the functions being called by the program to better understand it (refer to screenshot 14). I will be explaining my thought process in the next section, in it’s respective order.

Description of Learning Completed

When I began by conducting a similarity test the output of the tlsh.exe command can be explained as followed: the left column states the Core1Installer.exe, the middle column shows the name/hash of the malicious file, and the far-right column shows a number. This number represent the similarity of the inputted program compared to the others. For instance, the seventh row shows that .exe file has a value of 275. Since this value is the lowest value shown in the output that means that binary is somewhat similar. I say somewhat because 275 is a relatively high number. I’m primarily looking for a similar match of less than 20, which in that case, that would imply that the hashes are very similar.

My next step was to execute CAPA on the binary. CAPA, which stands for “Cyclical Analysis of Program Analysis”, is an open-source analysis tool designed for identifying capabilities in executables. It's commonly used in reverse engineering, malware analysis, and security research. With this program I’d be able to have a better grasp of the capabilities of the analyzed file. These are some simplified but significant findings I discovered of the binary based on the output: (1) The ATT&CK tactic and technique was shown to be defense evasion and discovery, and T1222 and T1083, respectively. (2) It uses HTTP communication. (3) It creates directories, read files and deletes them, and sets attributes. (4) It enumerates files on Windows, and finally, it contains a PDB file path.

When conducting the lookup of significant strings, I was able to pull out extremely valuable data. The first one I'd like to mention is the Windows file path, which as shown in the respective screenshot, it appears to be "E:\Core1Installer\Core1Installer\obj\Release\Core1Installer.pdb". For context, a .pdb file stands for "Program Database," and is a file format used in Microsoft Windows operating systems. It contains debugging and project state information used by debuggers and compilers. Then, my other significant finding is related with the ipv4 option I used previously. In the screenshot it clearly states an IPv4 address, which is **188.120.240.203**. This IP is an actual global routable address (public). I even did an IP lookup where it shows that this IP address is correlated to a particular geographic area, Russia (refer to screenshot 8). Based on the information shown in screenshot 8, it appears that this malware might've connect to an IP that is used in a VPN server.

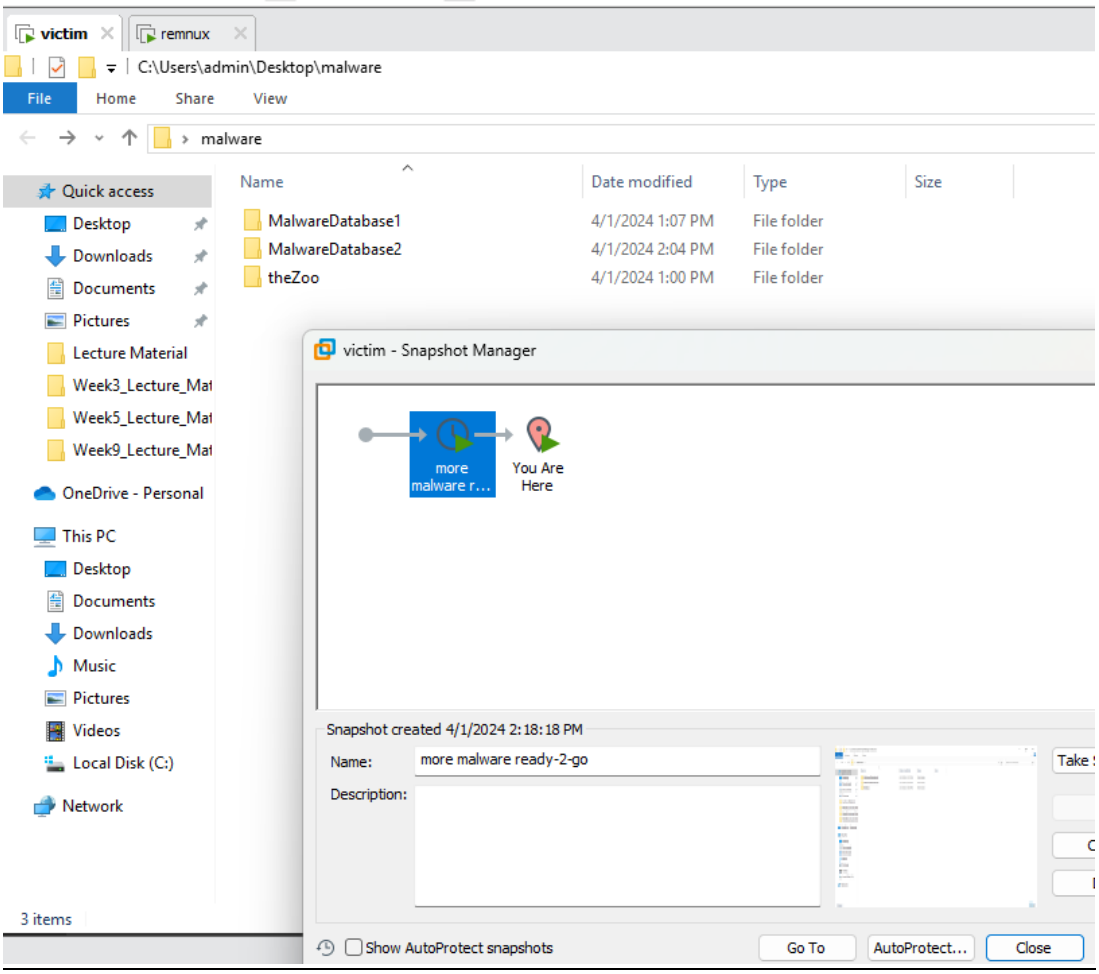
Then, I opened the PEid program to scan the file and it showed that the executable was more likely in C# language code and compiled using the .NET software framework that resides in the mscoree.dll. When I opened the PEstudio program to get even more data and went to the "Indicators" tab I noticed that the same IP address I found earlier was shown there. It also showed me some of the imports it uses (refer to screenshot 11 & 12). By now I believe it's common sense to think that this malware is more likely going to execute some sort of thing remotely or at least try to establish a connection somewhere else. Some of the imports I found valuable were: HttpRequest, WebResponse, WebRequest, WebHeaderCollection, DeflateStream, CompressionLevel, FilePath, FileNameFormat, ImageFormat, HideFiles, FolderName, and FolderPath.

Finally, I used the Ghidra program made by the NSA to understand the assembly language of the program. When I opened up the file, at first, I felt disoriented because of my lack of knowledge of assembly language. So, I had to depend on YouTube videos to gain as much knowledge as possible. After watching several videos, I could now say that at least I've gotten my feet wet with reverse engineering. While at the CodeBrowser in Ghidra, my first step was to look at the decompile code as shown in the stated screenshot. I had no clue what the variables meant, or even what the program did.

The only thing I did know was the structure of the code. The code starts with a function() X and inside the function it states the several variables it will use in the program. Then logic of the program follows if-else statements and try statements. I'm guessing that depending on the parameters used, if it's correct it will follow that path, if not, it will follow another path. It appears to have error-check logic because of the try statements. If the try statements execute correctly then it would stay there if not it would move onto the next try statement. Then, I looked at the functions used by the program. The ones that stood out to me and made more logic were: AsImageFormat, entry, FromBytes, textBox1_TextChanged, and Write. As of now, it appears that the pattern "image" had appeared in different instances of the program, as well as writing files and "bytes". Therefore, I believe this program deals with the modification, creation, or deletion of files, but more specifically images, within an operating system. It makes me believe that this program does the previously stated steps while at the same time establishing a connection with the IPv4 address that was found earlier. It is still not clear how this program behaves exactly, for that reason, a dynamic test would be needed. I will be conducting a dynamic analysis of the executable in milestone 4. Consequently, I will be disinfecting the computer and coming up with final conclusions during milestone 5.

Documentation of Work Completed

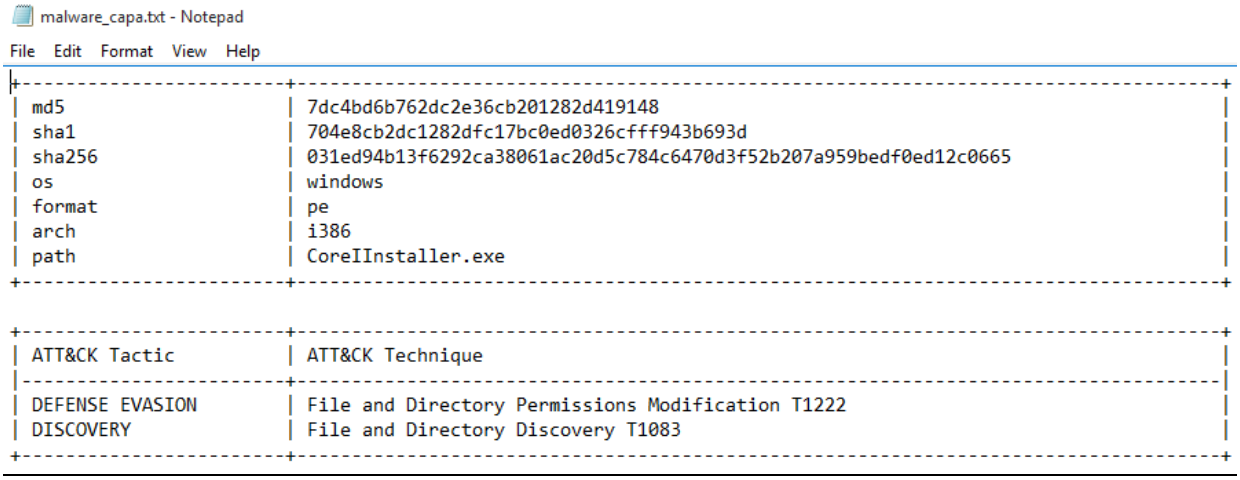
Screenshot 1



Screenshot 2

```
C:\Users\admin\Desktop>tlsh -c CoreIInstaller.exe -l malware_digest.txt
CoreIInstaller.exe      malware_samples/0002211144555787555111.msi      409
CoreIInstaller.exe      malware_samples/183b4e8853977551d6bf68ce7d1ecb386af257e1b2d6f954505d10413f2fe39d.dll    666
CoreIInstaller.exe      malware_samples/1c133b9bb476879df8145370ce1069ec92f28cade85a839e0159158a3e1b1afd.exe    289
CoreIInstaller.exe      malware_samples/1cf36a2d8a2206cb4758dcdbd0274f21e6f437079ea39772e821a32a76271d46.unknown
305
CoreIInstaller.exe      malware_samples/221209ea7151b399298c882daa26297c5b299f44369340d1050c82ff2b8865d8.msi    408
CoreIInstaller.exe      malware_samples/27b181f7f408f0f5f6f6d8f3d395fd499b47bce36044865d57fe10eaa2a440c2.unknown
633
CoreIInstaller.exe      malware_samples/2aaad8177d08507b09dc3d419b4d31f65db6fe6bc6314ffce650b9fc57f0817c.dll    663
CoreIInstaller.exe      malware_samples/41c9d28653704e628d8dd20e5f65a298242072156a31bc5fe0e24a1f4c640af5.exe    275
CoreIInstaller.exe      malware_samples/5a0daa24b5748d81ba0bb78d7f2b50eb4c387ffe679c92c1462f7dec586adb1f.ps1    505
CoreIInstaller.exe      malware_samples/651bc82076659431e06327aeb3aacef2c30bf3cfd43ae4f9bc6b4222f15bb673.unknown
360
CoreIInstaller.exe      malware_samples/686e60d6079a08eaafcdca5ab248cbc18cae7c6871b989c3bcbcb9a02fd5fad9.exe    751
CoreIInstaller.exe      malware_samples/6b4dd13ea6241a6c8ad2c967d88f3336798dc1e30dd24cfa3377f9b363d70b2e.exe    285
CoreIInstaller.exe      malware_samples/736cb1644adf35fc139e7031d9bc5073816784d21b34de83260387f47f13ba43.exe    377
CoreIInstaller.exe      malware_samples/7373bf246de45665456d475877db908aaf24047832483f8beff43e684c317305.unknown
547
CoreIInstaller.exe      malware_samples/817c226e42f5c503325288fd8273bc03b326590f457e7a589eb34c2792d0a5db.dll    498
CoreIInstaller.exe      malware_samples/93488eab403fafb3d8e10d38c80f0af745e3fa4cf26228acff24d35a149f6269.dll    644
CoreIInstaller.exe      malware_samples/MIT-MULTA9662778901.msi 408
CoreIInstaller.exe      malware_samples/MULTAMIT8069218371.msi  411
CoreIInstaller.exe      malware_samples/b2c0b8e5c095f109f7b030fbcbd5258e9e3d1cc6438d1a6562ce52827097bf11.unknown
664
CoreIInstaller.exe      malware_samples/bb8c0e477512adab1db26eb77fe10dadbc5dcbf8e94569061c7199ca4626a420.exe    484
CoreIInstaller.exe      malware_samples/c3a382298e7b40c769094035a49abe71314c89509848cd9485467c2179193a0f.msi    411
CoreIInstaller.exe      malware_samples/e24021c34cf961f2a17e8f5813e5be1240981f2c5ce5cacac3994c5dfb8cf077.unknown
630
CoreIInstaller.exe      malware_samples/f1b53f5353fa9ba6ddce5df301e28b5c68947f032463c220d163d03ab95832ef.msi    331
CoreIInstaller.exe      malware_samples/f5c245bd4d7eb95f9a2afde8960ef9c9640ad426a8e438b52caca1541b928954.exe    464
CoreIInstaller.exe      malware_samples/kx_qyu.msi      537
```

Screenshot 3



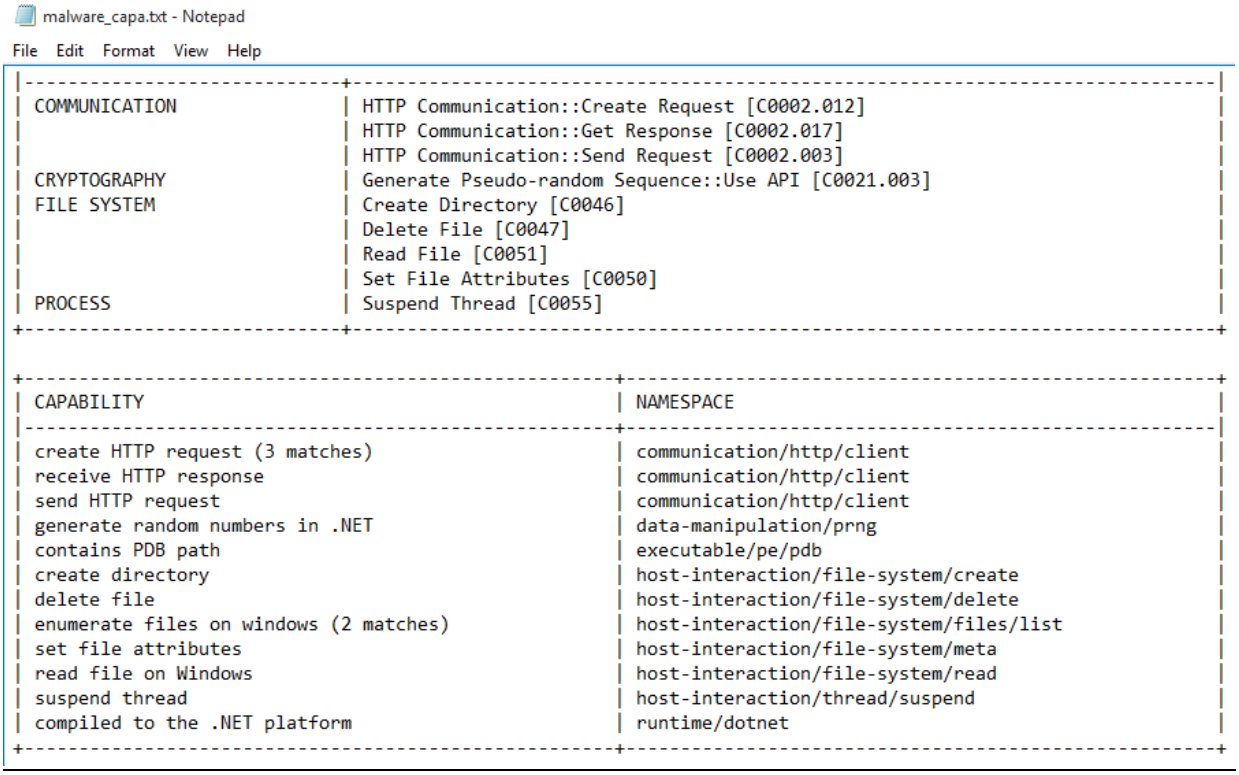
malware_capa.txt - Notepad

File Edit Format View Help

md5	7dc4bd6b762dc2e36cb201282d419148
sha1	704e8cb2dc1282dfc17bc0ed0326cfff943b693d
sha256	031ed94b13f6292ca38061ac20d5c784c6470d3f52b207a959bedf0ed12c0665
os	windows
format	pe
arch	i386
path	CoreIInstaller.exe

ATT&CK Tactic	ATT&CK Technique
DEFENSE EVASION	File and Directory Permissions Modification T1222
DISCOVERY	File and Directory Discovery T1083

Screenshot 4



malware_capa.txt - Notepad

File Edit Format View Help

COMMUNICATION	HTTP Communication::Create Request [C0002.012]
	HTTP Communication::Get Response [C0002.017]
	HTTP Communication::Send Request [C0002.003]
CRYPTOGRAPHY	Generate Pseudo-random Sequence::Use API [C0021.003]
FILE SYSTEM	Create Directory [C0046]
	Delete File [C0047]
	Read File [C0051]
	Set File Attributes [C0050]
PROCESS	Suspend Thread [C0055]

CAPABILITY	NAMESPACE
create HTTP request (3 matches)	communication/http/client
receive HTTP response	communication/http/client
send HTTP request	communication/http/client
generate random numbers in .NET	data-manipulation/prng
contains PDB path	executable/pe/pdb
create directory	host-interaction/file-system/create
delete file	host-interaction/file-system/delete
enumerate files on windows (2 matches)	host-interaction/file-system/files/list
set file attributes	host-interaction/file-system/meta
read file on Windows	host-interaction/file-system/read
suspend thread	host-interaction/thread/suspend
compiled to the .NET platform	runtime/dotnet

Screenshot 5

Command Prompt

```
C:\Users\admin\Desktop>bstrings.exe -p
Name            Description
aeon            Finds Aeon wallet addresses
b64             Finds valid formatted base 64 strings
bitcoin         Finds BitCoin wallet addresses
bitlocker       Finds Bitlocker recovery keys
bytecoin        Finds ByteCoin wallet addresses
cc              Finds credit card numbers
dashcoin        Finds DashCoin wallet addresses (D*)
dashcoin2       Finds DashCoin wallet addresses (7|X)*
email           Finds embedded email addresses
fantomcoin      Finds Fantomcoin wallet addresses
guid            Finds GUIDs
ipv4            Finds IP version 4 addresses
ipv6            Finds IP version 6 addresses
mac             Finds MAC addresses
monero          Finds Monero wallet addresses
reg_path        Finds paths related to Registry hives
sid             Finds Microsoft Security Identifiers (SID)
ssn             Finds US Social Security Numbers
sumokoin        Finds SumoKoin wallet addresses
unc             Finds UNC paths
url3986         Finds URLs according to RFC 3986
urlUser         Finds usernames in URLs
usPhone         Finds US phone numbers
var_set         Finds environment variables being set (OS=Windows_NT)
win_path        Finds Windows style paths (C:\folder1\folder2\file.txt)
xml             Finds XML/HTML tags
zip             Finds zip codes
```

Screenshot 6

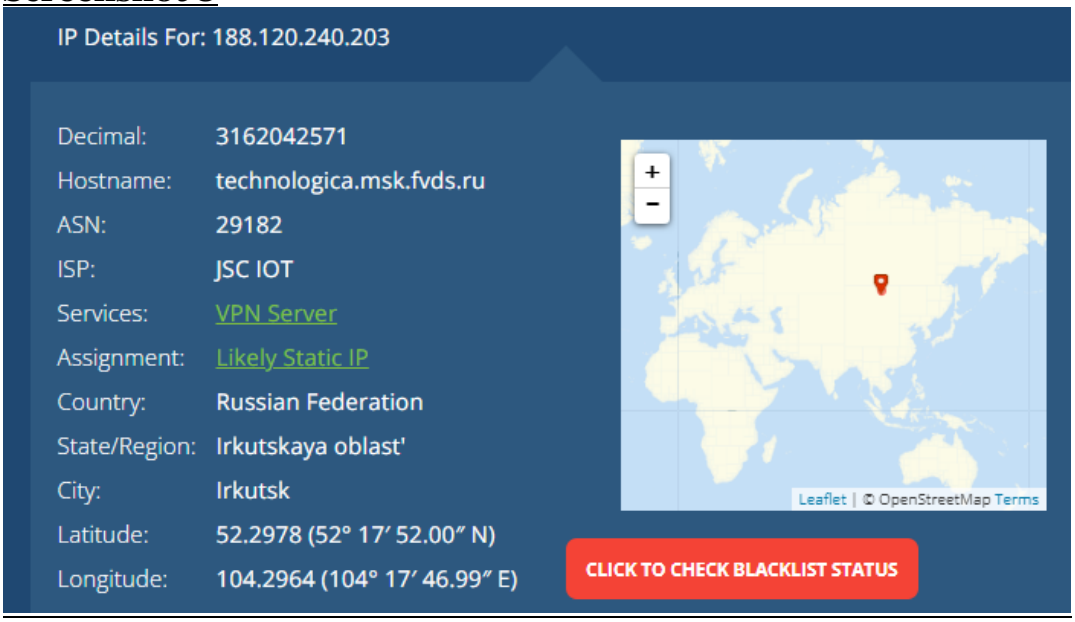
Command Prompt

```
C:\Users\admin\Desktop>bstrings.exe -f CoreIInstaller.exe --lr ipv4
bstrings version 1.5.1.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/bstrings

Command line: -f CoreIInstaller.exe --lr ipv4

Searching via RegEx pattern: \b(?:(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])\.){3}(?:25[0-5]|2[0-4][0-9]|1[0-9][0-9
]|[1-9]?[0-9])\b

Searching 1 chunk (512 MB each) across 420 KB in 'C:\Users\admin\Desktop\CoreIInstaller.exe'

Chunk 1 of 1 finished. Total strings so far: 2,040 Elapsed time: 0.166 seconds. Average strings/sec: 12,286
Primary search complete. Looking for strings across chunk boundaries...
Search complete.

Processing strings...

188.120.240.203
0.0.0.5
4.0.0.0
11.0.0.0
lSystem.Resources.ResourceReader, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089#System.Res
ources.RuntimeResourceSet
fSystem.Drawing.Icon, System.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3aBj
QSystem.Drawing, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
<assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>

Found 8 strings in 0.172 seconds. Average strings/sec: 11,871
```

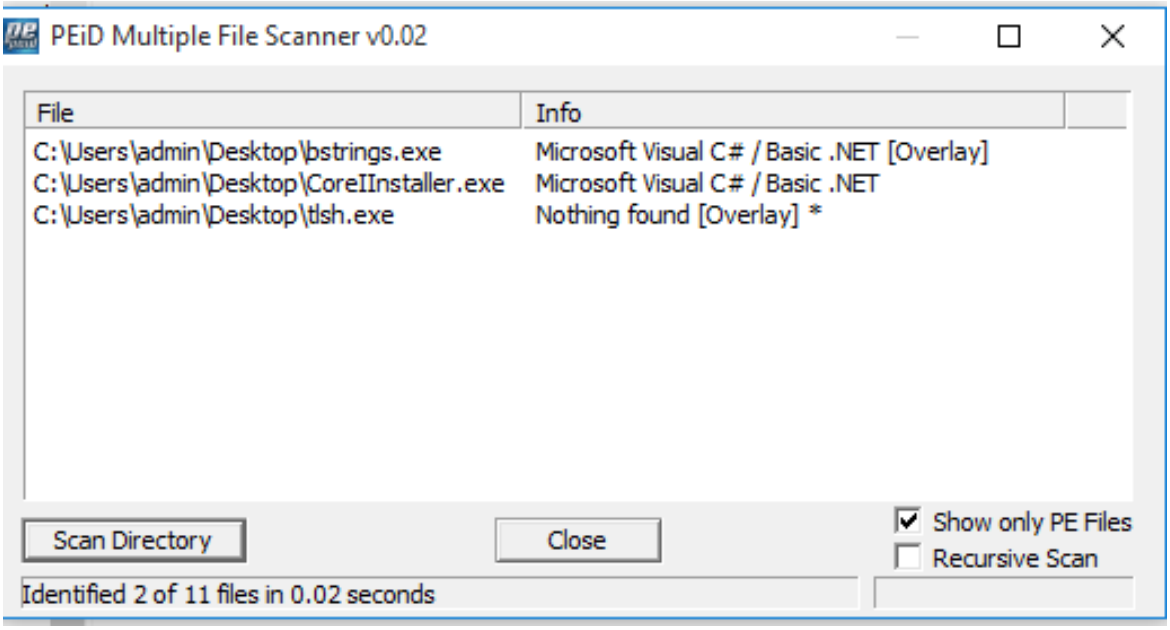
Screenshot 7

```
Command Prompt
Command line: -f CoreIInstaller.exe --lr win_path

Searching via RegEx pattern: (?:"?[a-zA-Z]\:|\\\\[^\\\/\:\*\?\<\>\|]+\\[^\\\/\:\*\?\<\>\|]*)\\
*\w([^\\\/\:\*\?\<\>\|])*

Searching 1 chunk (512 MB each) across 420 KB in 'C:\Users\admin\Desktop\CoreIInstaller.exe'

Chunk 1 of 1 finished. Total strings so far: 2,040 Elapsed time: 0.165 seconds. Average string
Primary search complete. Looking for strings across chunk boundaries...
Search complete.

Processing strings...

tO:\B
E:\CorelInstaller\CorelInstaller\obj\Release\CorelInstaller.pdb

Found 2 strings in 0.172 seconds. Average strings/sec: 11,855

C:\Users\admin\Desktop>bstrings.exe -f CoreIInstaller.exe --lr var_set
bstrings version 1.5.1.0

Author: Eric Zimmerman (saericzimmerman@gmail.com)
https://github.com/EricZimmerman/bstrings

Command line: -f CoreIInstaller.exe --lr var_set

Searching via RegEx pattern: ^[a-z_0-9]+=[\\/:\*\?<>|;\- _a-z0-9]+

Searching 1 chunk (512 MB each) across 420 KB in 'C:\Users\admin\Desktop\CoreIInstaller.exe'
```

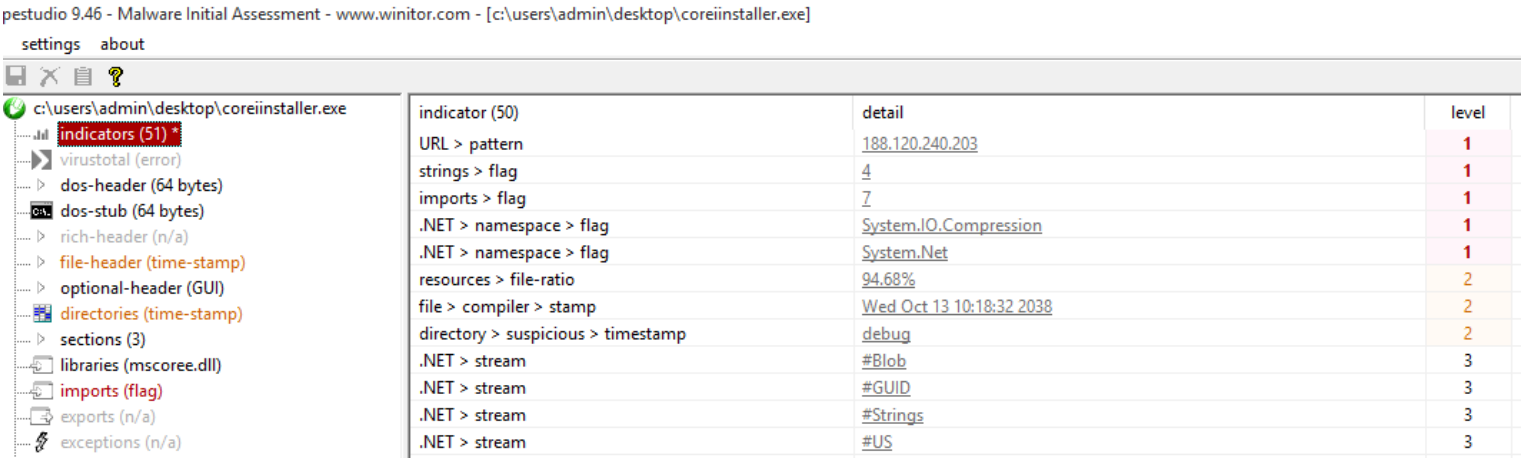
Screenshot 8



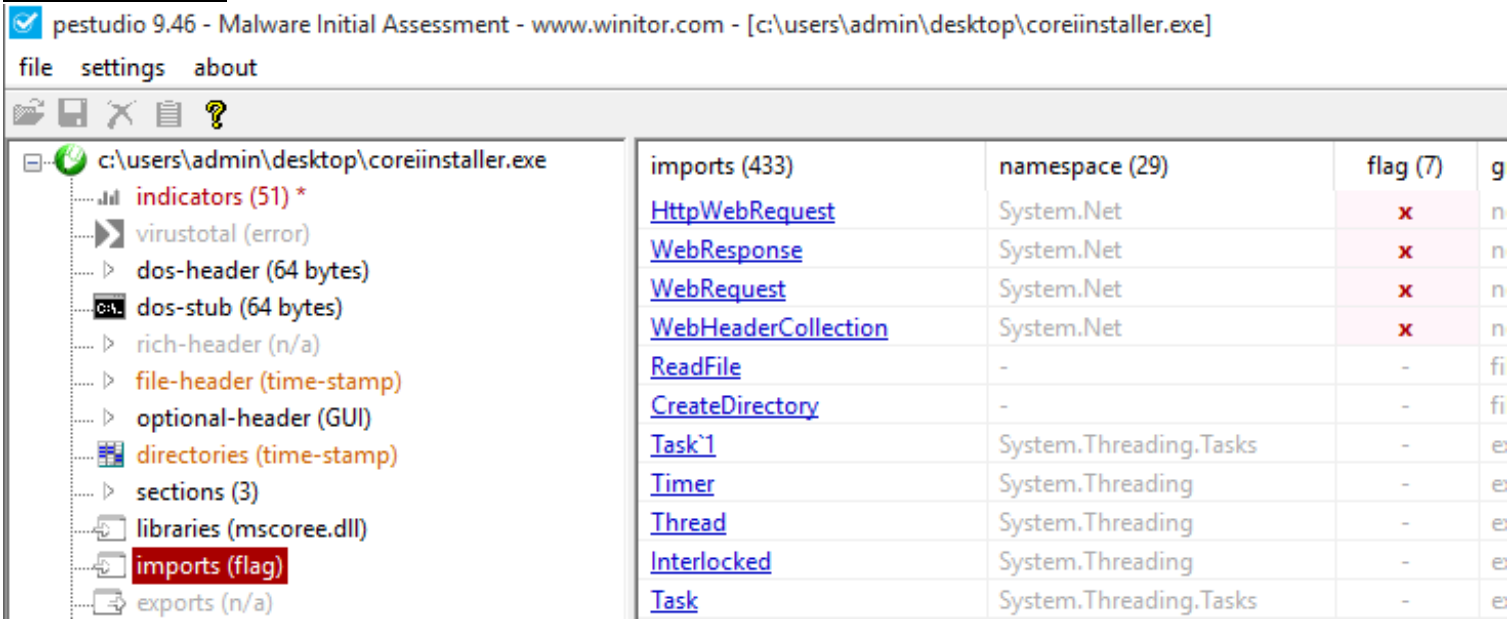
Screenshot 9



Screenshot 10

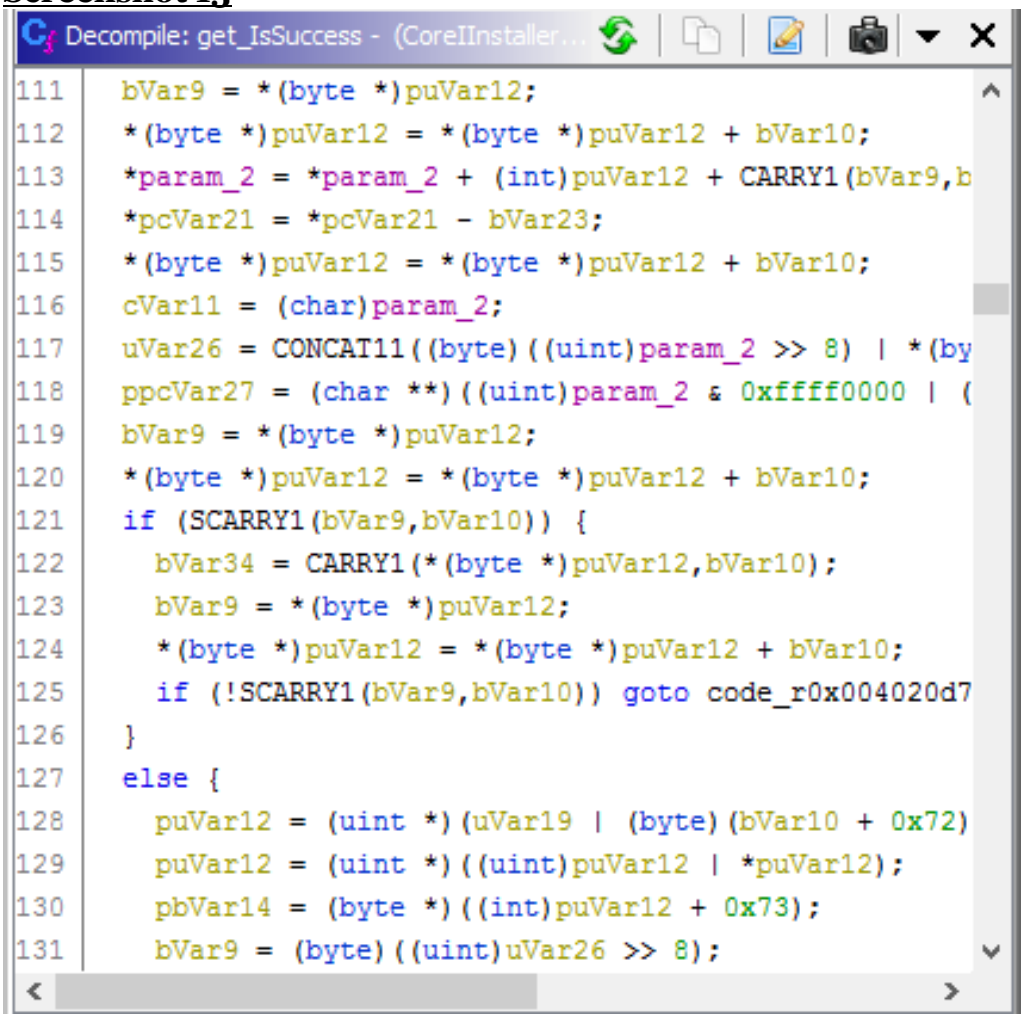


Screenshot 11

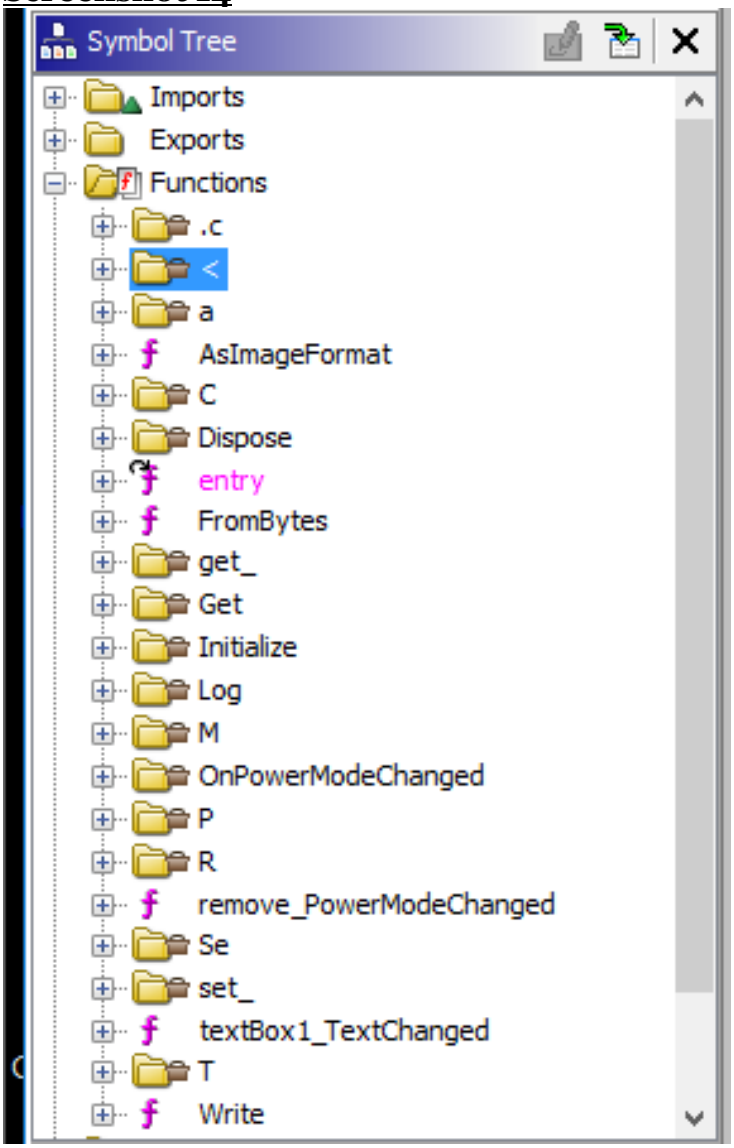


Screenshot 12

Screenshot 13



Screenshot 14



References

Low Level Learning. (2023, January 14). *Everything is open source if you can reverse engineer (try it right now!)*. YouTube. https://www.youtube.com/watch?v=gh2RXE9BIN8

stacksmashing. (2019, March 8). *Ghidra Quickstart & Tutorial: Solving a simple crackme*. YouTube. https://www.youtube.com/watch?v=fTGTnrgjuGA