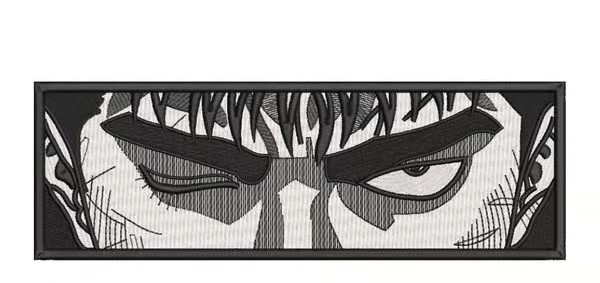
****  
  
  
  
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Name: Saifullah Anwar

Roll No: 231308

Subject: Malware Analysis

Class: BS-CYS-F23-A

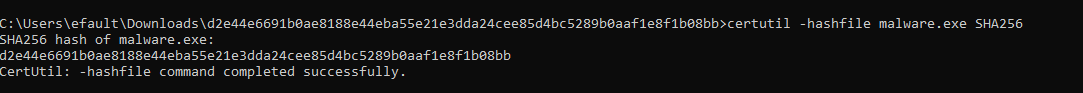
Task: 2

PROLOGUE:

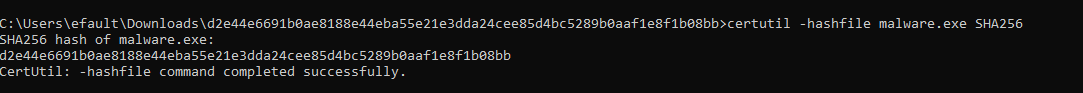
In this lab we will conduct the static Analysis on Malware. Random malware has been downloaded on the windows VM. The prologue is about getting hashes of the malware whether md5 or sha256. We will see how one can generate the hash which is important for further analysis in Virus Total. There are many ways for the hash generation but the ways that I know are as given below.

Cert-util:

There is a utility for windows which is used to display and dump the CA configuration information. We won’t talk about CA info here but it’s a popular utility which is being abused by the black hats to download or De-obfuscate the malicious files. You can generate hashes too by using this utility as shown in the below screenshot.

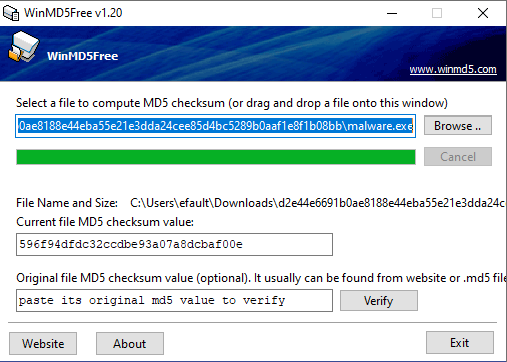


Above screenshot is all about generating SHA256 hash for the file that we are analyzing. The same utility can be used to generate the MD5 hash too as shown   
in below screenshot.



Winmd5:

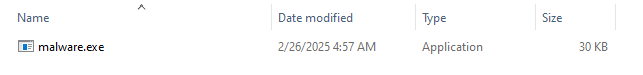
Now there is another program a bit old but the goated one in generating the MD5 hashes only. Refer to the below screenshot.



Note:  
 *“However there is one thing to be noted that I just closed my machine so I forgot to add the hash to Virus-total. It is a fact that the file we are analyzing is the malicious file. So, obviously the engines of Virus-Total will raise the file as malicious”.*  
  
1st STEP:

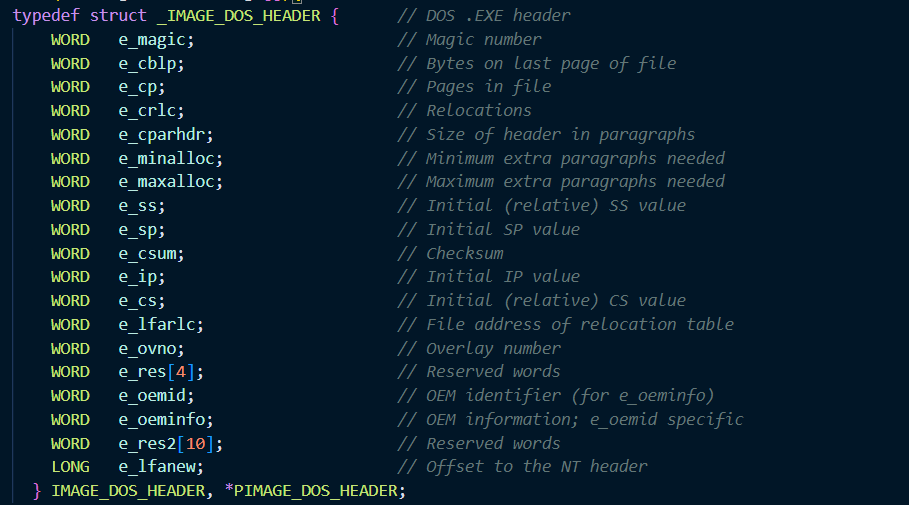
So, we got our malware from [Malware Bazaar](https://bazaar.abuse.ch/), time to analyze the dawg stuff going on inside this file.

You can refer to it, as shown in the below screenshot.

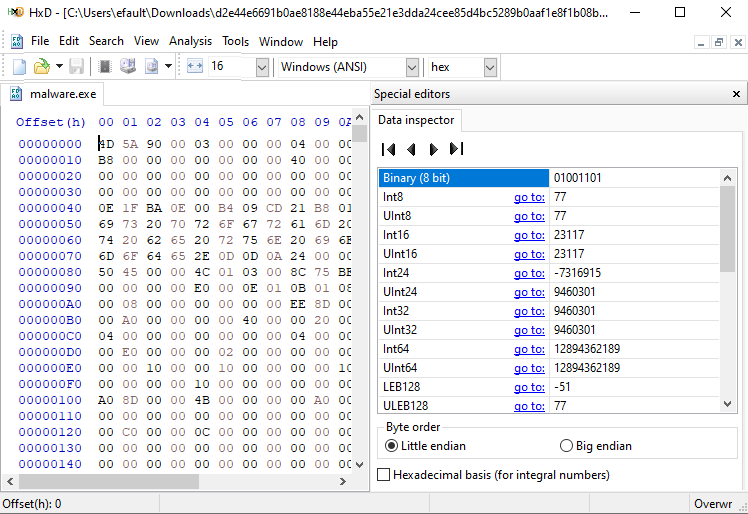


I don’t know what kind of malware it is 😉 but let’s see what happens!

From the bird’s eye view we know that it is Portable Executable (PE) file. Before moving on to the magic header here is the basic structure of \_IMAGE\_DOS\_HEADER shown below.



The above structure is the definition for \_IMAGE\_DOS\_HEADER which consists of several fields. We will not explain every field but for now the only concerning field for us is WORD e\_magic. Which tells about the magic number of the file that we are currently working on. Magic number is basically the identification number for a file. To analyze the magic number of the executable we will use the program call [Hxd](https://mh-nexus.de/en/hxd/) or known as the hex-editor. Just load the binary into Hxd and observe the Magic number.

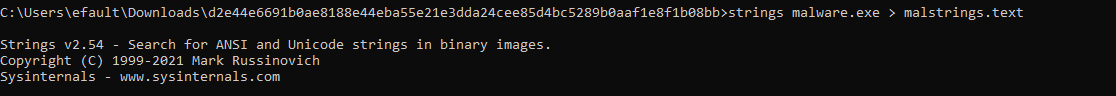
   
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
As shown in the first entity (0x00000000) we have 4D 5A or 0x5A4D and is the hexadecimal representation of ASCII characters MZ, which is by default clear that it is a Portable Executable File. The other entity is e\_lfnew which contains the offset to the NT headers. They contain a lot of information which is related to the PE file.

2nd STEP:

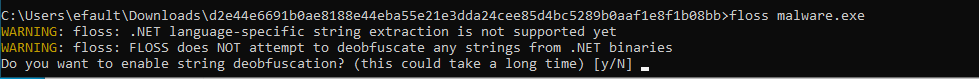
The next step is to look for string inside the executable. So again, there are many ways for it. Some of them are as given below.

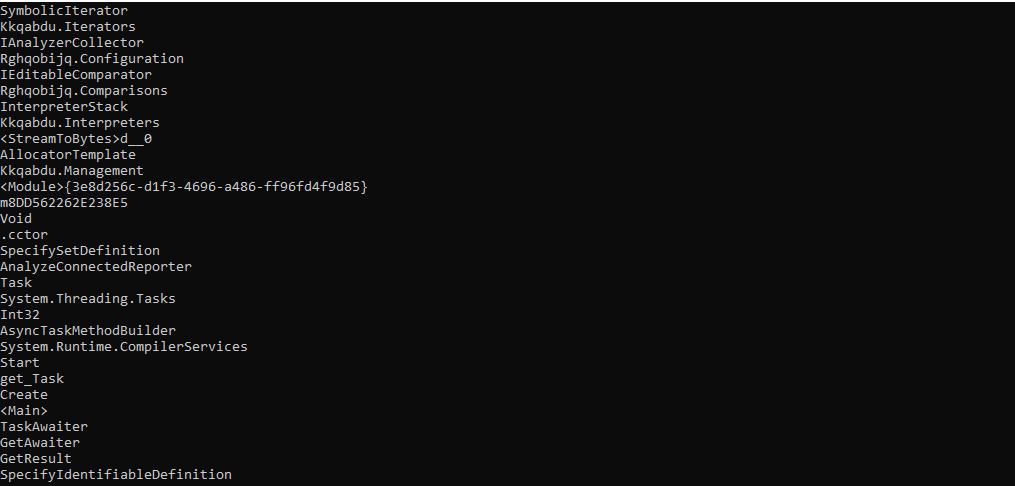
Strings:

Now time for the first utility which is strings that is already present in Sys-Internal suite if I am not forgetting. I have just redirected the output to the file so the 2nd screenshot is the result from the string command.

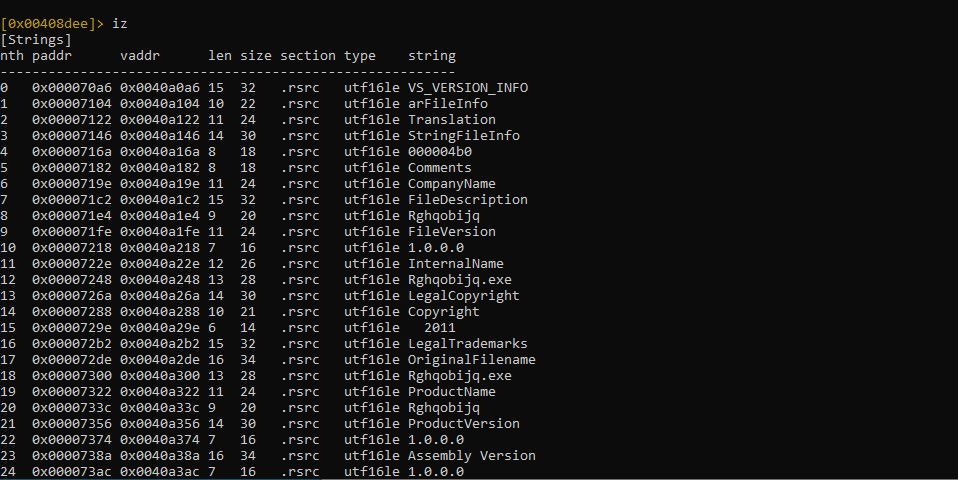
  
  
Floss:

Floss is the utility that is based on python and its absolutely amazing one. Reference to the below screenshot.

  
  
Hardcoded strings 😉. By the way let’s look for another screenshot.

Radare2:

Radare2 is one of the most powerful debuggers and reversing tools out there. So here we used radare2 to extract out the strings using (iz) as shown in below screenshot.

  
  
  
  
  
  
  
  
  
Note:

*“As you can see in the screenshot the (.RSRC) is the section in PE file which contains resources used by the application such as icons, images, menus and especially strings.”*

Here we got our strings from (.RSRC) section as shown in the above snippet.

ADDITIONAL STEP:  
There was some sort of additional step. We loaded the binary in radare2 use (aaa) command to analyze the binary from start to end. Now we used (ii) command to look up for the import functions.

Imported Functions:

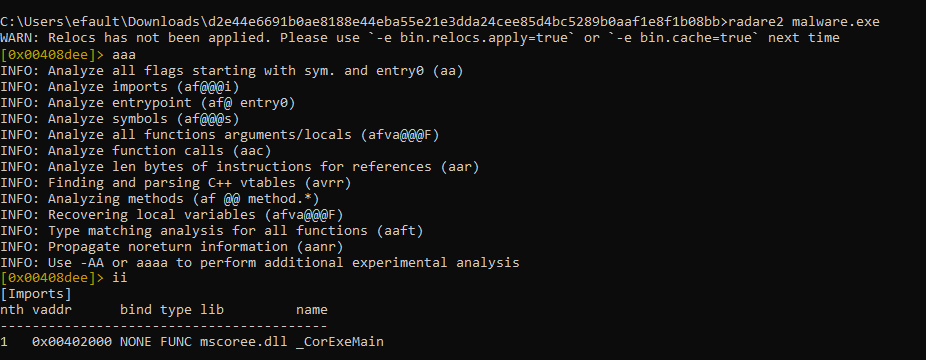
Imported functions are those functions, that are imported from the external libraries meaning they are not used here in the program but called externally.

Continued...

In the below screenshot radare2 is showing the imported function and same thing was seen in IDA both debuggers/disassemblers showed the imported function which is mscoree.dll.

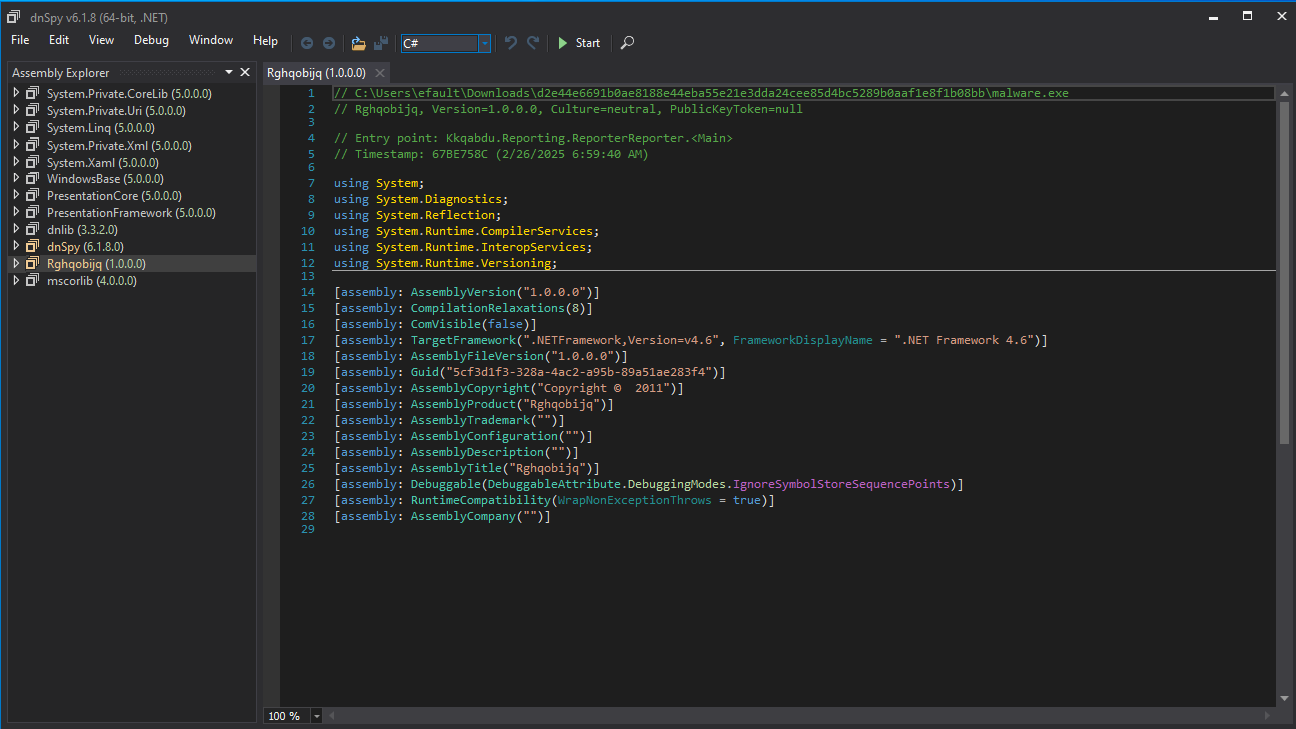
According to the community online which states that.

“If the import table includes mscoree.dll and include the entry for the function \_CoreExeMain then definitely the exe is .NET compiled”

And same thing is happening in the below screenshot.I am not explaining the why and what’s of these imports for several reason so for now just remember the binary is .NET compiled.

.NET Analysis:

If you want to analyze the .NET or C# binaries, there are many cool tools out there but I would suggest just go for DNSPY, one of the beautiful tools out there you can just download it from there official Git-hub repo. Refer to the below screenshot.



We loaded the binary into it and see what we got, A more formattable way to analyze the stuff. Many headers are used here but we will not explain here. So, stopping the static analysis here because still there is stuff related to .NET that I didn’t cover.

-----------------------------------------------------------------------------