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.

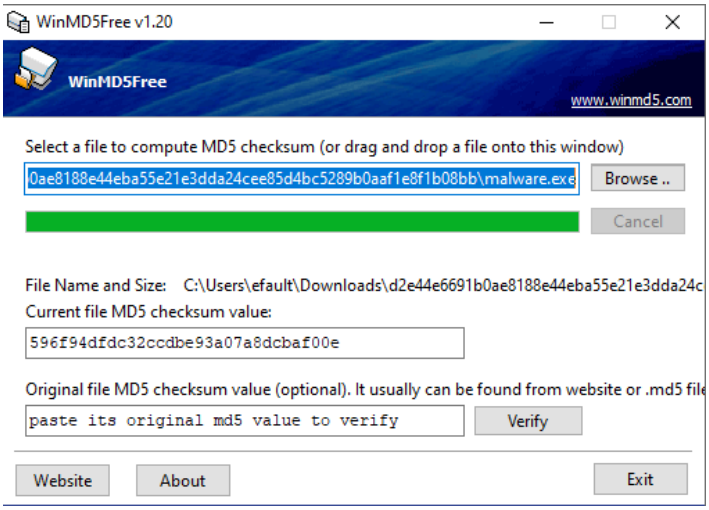
```
C:\Users\efault\Downloads\d2e44e6691b0ae8188e44eba55e21e3dda24cee85d4bc5289b0aaf1e8f1b08bb>certutil -hashfile malware.exe SHA256
SHA256 hash of malware.exe:
d2e44e6691b0ae8188e44eba55e21e3dda24cee85d4bc5289b0aaf1e8f1b08bb
CertUtil: -hashfile command completed successfully.
```

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.

```
C:\Users\efault\Downloads\d2e44e6691b0ae8188e44eba55e21e3dda24cee85d4bc5289b0aaf1e8f1b08bb>certutil -hashfile malware.exe SHA256
SHA256 hash of malware.exe:
d2e44e6691b0ae8188e44eba55e21e3dda24cee85d4bc5289b0aaf1e8f1b08bb
CertUtil: -hashfile command completed successfully.
```

Winmd5:

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



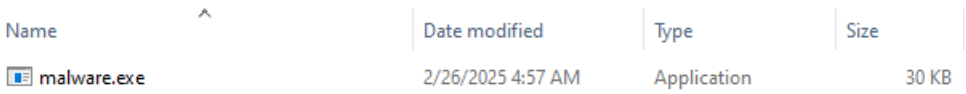
Note:

"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, time to analyze the dawg stuff going on inside this file.

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

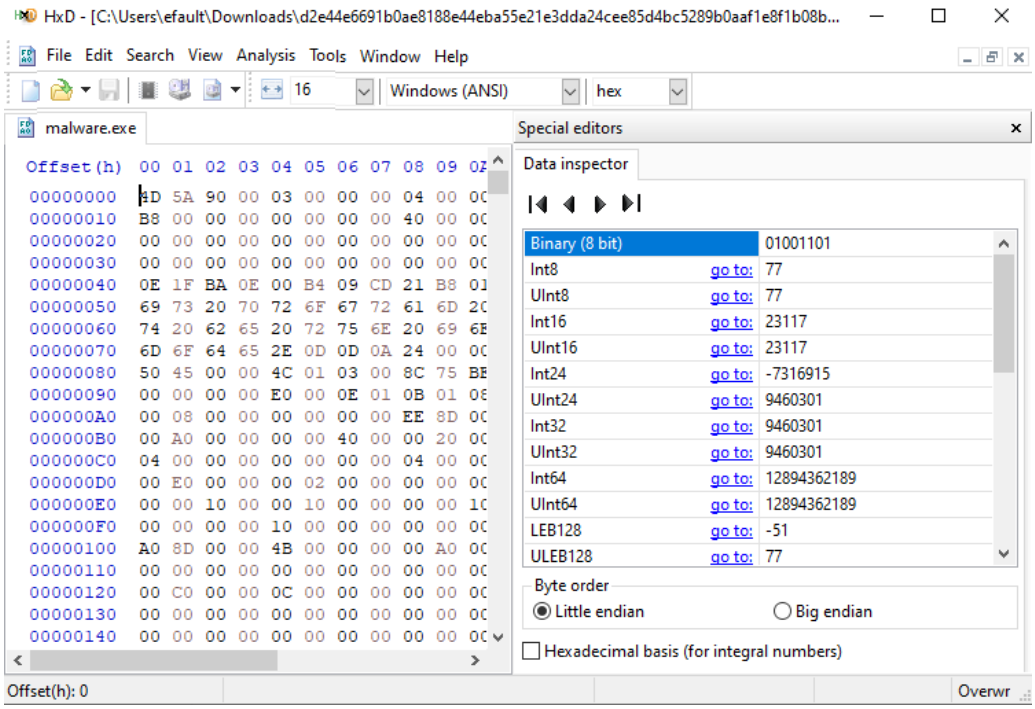


I don't know what kind of malware it is 😉 but let's see what happens!

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.

```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
    WORD   e_magic;                     // Magic number
    WORD   e_cblp;                      // Bytes on last page of file
    WORD   e_cp;                        // Pages in file
    WORD   e_crlc;                      // Relocations
    WORD   e_cparhdr;                   // Size of header in paragraphs
    WORD   e_minalloc;                  // Minimum extra paragraphs needed
    WORD   e_maxalloc;                  // Maximum extra paragraphs needed
    WORD   e_ss;                        // Initial (relative) SS value
    WORD   e_sp;                        // Initial SP value
    WORD   e_csum;                      // Checksum
    WORD   e_ip;                        // Initial IP value
    WORD   e_cs;                        // Initial (relative) CS value
    WORD   e_lfarlc;                    // File address of relocation table
    WORD   e_ovno;                      // Overlay number
    WORD   e_res[4];                    // Reserved words
    WORD   e_oemid;                     // OEM identifier (for e_oeminfo)
    WORD   e_oeminfo;                   // OEM information; e_oemid specific
    WORD   e_res2[10];                  // Reserved words
    LONG   e_lfanew;                    // Offset to the NT header
  } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

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 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.

```
C:\Users\efault\Downloads\d2e44e6691b0ae8188e44eba55e21e3dda24cee85d4bc5289b0aaf1e8f1b08bb>strings malware.exe > malstrings.text

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com
```

Floss:

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

```
C:\Users\efault\Downloads\d2e44e6691b0ae8188e44eba55e21e3dda24cee85d4bc5289b0aaf1e8f1b08bb>floss malware.exe
WARNING: floss: .NET language-specific string extraction is not supported yet
WARNING: floss: FLOSS does NOT attempt to deobfuscate any strings from .NET binaries
Do you want to enable string deobfuscation? (this could take a long time) [y/N] _
```

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

```
SymbolicIterator
Kkqabdu.Iterators
IAnalyzerCollector
Rghqobijq.Configuration
IEditableComparator
Rghqobijq.Comparisons
InterpreterStack
Kkqabdu.Interpreters
<StreamToBytes>d__0
AllocatorTemplate
Kkqabdu.Management
<Module>{3e8d256c-d1f3-4696-a486-ff96fd4f9d85}
m8DD562262E238E5
Void
.cctor
SpecifySetDefinition
AnalyzeConnectedReporter
Task
System.Threading.Tasks
Int32
AsyncTaskMethodBuilder
System.Runtime.CompilerServices
Start
get_Task
Create
<Main>
TaskAwaiter
GetAwaiter
GetResult
SpecifyIdentifiableDefinition
```

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.

```
[0x00408dee]> iz
[Strings]
nth paddr      vaddr      len size section type    string
-------------------------------------------------------------
0   0x000070a6 0x0040a0a6 15  32   .rsrc   utf16le VS_VERSION_INFO
1   0x00007104 0x0040a104 10  22   .rsrc   utf16le arFileInfo
2   0x00007122 0x0040a122 11  24   .rsrc   utf16le Translation
3   0x00007146 0x0040a146 14  30   .rsrc   utf16le StringFileInfo
4   0x0000716a 0x0040a16a 8   18   .rsrc   utf16le 000004b0
5   0x00007182 0x0040a182 8   18   .rsrc   utf16le Comments
6   0x0000719e 0x0040a19e 11  24   .rsrc   utf16le CompanyName
7   0x000071c2 0x0040a1c2 15  32   .rsrc   utf16le FileDescription
8   0x000071e4 0x0040a1e4 9   20   .rsrc   utf16le Rghqobijq
9   0x000071fe 0x0040a1fe 11  24   .rsrc   utf16le FileVersion
10  0x00007218 0x0040a218 7   16   .rsrc   utf16le 1.0.0.0
11  0x0000722e 0x0040a22e 12  26   .rsrc   utf16le InternalName
12  0x00007248 0x0040a248 13  28   .rsrc   utf16le Rghqobijq.exe
13  0x0000726a 0x0040a26a 14  30   .rsrc   utf16le LegalCopyright
14  0x00007288 0x0040a288 10  21   .rsrc   utf16le Copyright
15  0x0000729e 0x0040a29e 6   14   .rsrc   utf16le   2011
16  0x000072b2 0x0040a2b2 15  32   .rsrc   utf16le LegalTrademarks
17  0x000072de 0x0040a2de 16  34   .rsrc   utf16le OriginalFilename
18  0x00007300 0x0040a300 13  28   .rsrc   utf16le Rghqobijq.exe
19  0x00007322 0x0040a322 11  24   .rsrc   utf16le ProductName
20  0x0000733c 0x0040a33c 9   20   .rsrc   utf16le Rghqobijq
21  0x00007356 0x0040a356 14  30   .rsrc   utf16le ProductVersion
22  0x00007374 0x0040a374 7   16   .rsrc   utf16le 1.0.0.0
23  0x0000738a 0x0040a38a 16  34   .rsrc   utf16le Assembly Version
24  0x000073ac 0x0040a3ac 7   16   .rsrc   utf16le 1.0.0.0
```

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"

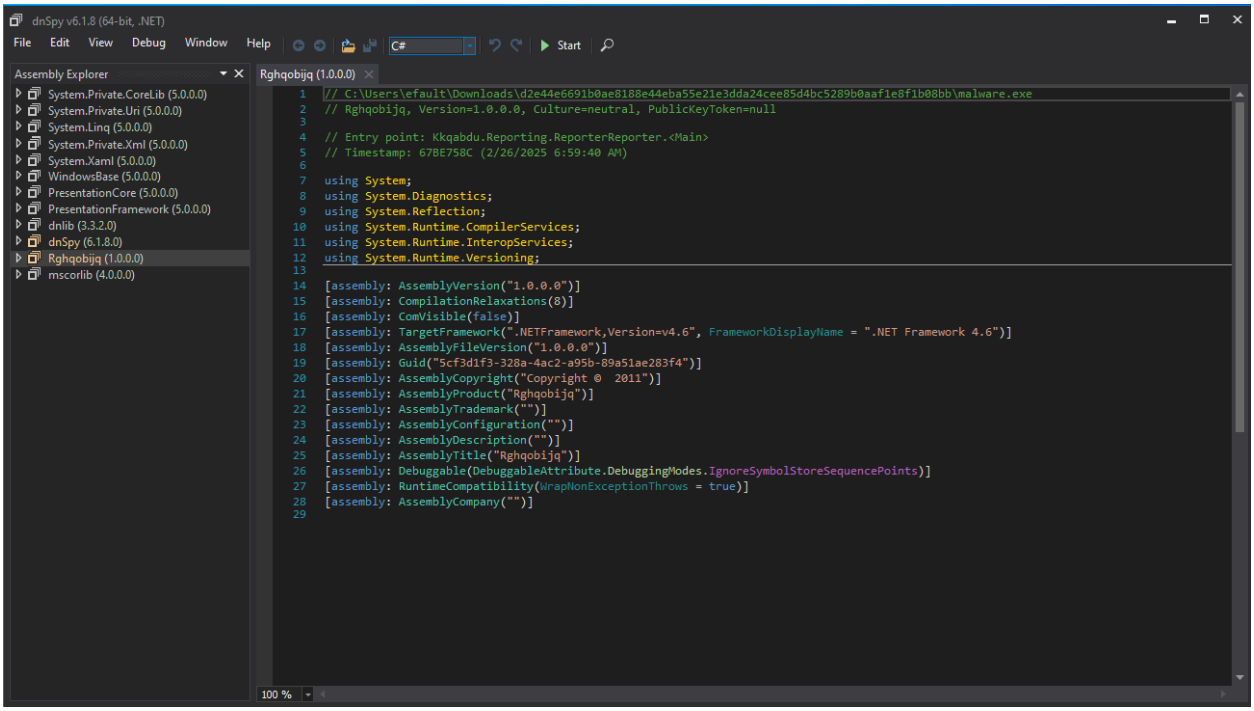
And same thing is happening in the below screenshot.

```
C:\Users\efault\Downloads\d2e44e6691b0ae8188e44eba55e21e3dda24cee85d4bc5289b0aaf1e8f1b08bb>radare2 malware.exe
WARN: Relocs has not been applied. Please use `-e bin.relocs.apply=true` or `-e bin.cache=true` next time
[0x00408dee]> aaa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze imports (af@@@i)
INFO: Analyze entrypoint (af@ entry0)
INFO: Analyze symbols (af@@@s)
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Analyzing methods (af @@ method.*)
INFO: Recovering local variables (afva@@@F)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Use -AA or aaaa to perform additional experimental analysis
[0x00408dee]> ii
[Imports]
nth vaddr      bind type lib        name
----------------------------------------
1   0x00402000 NONE FUNC mscoree.dll _CorExeMain
```

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.
