DE MONTFORT UNIVERSITY LEICESTER

Malware Analysis Report

A Report Covering the Basic, Static & Dynamic Analysis, as well as Reverse Engineering & Network Analysis of Potentially Malicious Malware.

Word Count: 3090

Contents

Table of Figures

1 - Introduction

This report details a comprehensive analysis of several malware samples. I have utilised the many tools provided by CTEC3754 in the Malware Module. I have placed the questions in Index 5 and referenced the questions in each section pertaining to the set of questions. I have made use of the lab and lecture materials as well seeking external references to condense my learning. ("Analysis of Malicious Documents- Part 5," n.d.), ("11 Best Malware Analysis Tools and Their Features," n.d.) and (Arshad, 2021)

1.1 – Scenario

The task involves retrieving and analysing PDF documents from the "cw_pdf_files.7z" archive file to determine if they are malicious. A comprehensive analysis is required, and findings are documented in the report. Additionally, the "unknown.file" from the "unknown.7z" archive file has been confirmed for its file type, .exe, and executed for analysis, and observations documented using appropriate tools. Basic static analysis, including analysing imports, exports, strings, and suspicious sections, have been performed, and the sample unpacked. Extensive dynamic analysis using tools like RegShot, Process Monitor, and CaptureBat has been conducted, and any observed changes, such as dropped, executed, or deleted files, is documented. Observable network activities of the malware that has been analysed and documented in both isolated and online environments. Reverse Engineering was also attempted on 'malsample.dll'.

1.2 – Environment & Tools

1.2.1 – Environment

NB: This section is providing the answer to question located in index 5.2.1 question 3(a) and index 5.1.1

I have opted to complete my analysis in the WindowsXP Virtual Machine (VM) provided by CTEC3754 as this is the best environment to analyse potentially malicious files in my experience and provides a barrier of security while enabling comprehensive analysis of potential malware. Should it get to a point where I need to utilise the REMnux VM I will endeavour to do so. ("REMnux Usage Tips for Malware Analysis on Linux," n.d.) I understand that REMnux is designed solely for the purpose of Malware Analysis. ("REMnux," n.d.)

Initially I had trouble setting up REMnux share folders. With the help of CTEC3754 I managed to download the files over the internet directly into the machine. Despite this, I struggled still as the tools were not installed, I attempted to set up Cuckoo Sandbox on Kali linux in virtual box and followed numerous guides but to no avail. ("Setting up Cuckoo Sandbox Step by Step Guide(Malware Analysis Tool) | by Lahiru Oshara Hinguruduwa | Medium," n.d.) I opted to attempt malware analysis within a Kali VM environment to see what information I could get this way. ("Setting Up A Kali Linux VM For Malware Analysis – Systran Box," n.d.)

1.2.2 - Analysis Tools

These are the tools used through my Basic and Dynamic Static Analysis, and Reverse Engineering (Security, 2022):

VirusTotal	Wireshark
Peepdf	FakeNet
PEiD	RegShot
PDFiD	MiTec EXE Explorer
PDFWalker	PEview
Procmon	IDAPro

Table.1.2.2.1 – Tools used during analysis stage

1.3 – Tasks & Questions

1.3.1 – Analysis Questions & Tasks Part 1

1.3.1.1 – PDF

Retrieve the two PDF documents from the "cw_pdf_files.7z" archive file. Perform a comprehensive analysis of the two files and present your findings, drawing conclusions as to whether or not each of the files may be a malicious PDF document.

After conducting a thorough analysis of the two PDF documents, it can be inferred that sample 1 does not display any malicious attributes or activity. However, sample 2 exhibits indications of obfuscation, a frequently used technique by malicious actors to conceal their intentions. The file also contains javascript and an embedded item, both of which are well-known elements of malicious PDF files. Further examination with tools such as PeePDF, PDFid, VirusTotal, PDF-Parser, and Wireshark suggests that sample 2 is probably a malicious file.

Upon opening sample 2, Wireshark reveals a flurry of activity, which is a tell-tale indication of a malicious PDF file. PDF-Parser has detected an embedded item inside the PDF file, often employed to execute malicious code. PeePDF and PDFid flagged the file for containing obfuscated code and javascript, respectively, both commonly present in malicious PDF files.

Consequently, although sample 1 seems to be a secure PDF file, sample 2 displays several characteristics commonly related to malicious PDF files. Therefore, it is advised that caution is exercised when handling sample 2, and that proper security precautions are taken before opening the file.

PLEASE SEE INDEX 2.1.2 FOR DETAILED INFORMATION & EVIDENCES.

1.3.1.2 – 'unknown.7z'

1. Retrieve "unknown.file" from the archive zipped file unknown.7z. (a) How would you confirm the type of file it is, and how will you make it execute for analysis? (b) Is the sample packed? What observable features of the file suggests that it may/may not be packed? Document all your observations with any applicable tools of your choice.

In summary of the basic and dynamic static analysis conducted, a variety of malware analysis tools were utilised to identify that the unknown file is likely packed and possibly contains malware. The analysis aided in the detection of abnormal behaviour, suspicious signatures, inconsistencies with legitimate files, and other indicators that suggest the file may be malicious in nature. Upon initial inspection, the file appeared to be an image but was discovered to be an executable file.

After retrieving the "unknown.file" sample, a range of tools were used to confirm the file type and analyse the sample. Firstly, VirusTotal was utilised to scan the file and various file extensions were tried to confirm that it was an executable file (.exe).

Subsequently, the sample was executed on a virtual machine to prevent any potential damage to the host system. Dynamic analysis tools were employed to observe the behaviour of the file upon execution.

During the analysis, it was discovered that the "unknown.file" sample was packed and employed obfuscation techniques to evade detection. The packer used was identified as UPX using PEiD. The analysis of the headers and sections of the executable file using PEView indicated that the file was compressed and had sections that were not readable by traditional PE viewers.

Further analysis using Filealyzer and MitecEXE revealed that the file had a high entropy value, which is an indicator of packing. Additionally, the file had a compressed size that was significantly smaller than its actual size, another indication of packing. The analysis confirmed that the "unknown.file" sample was indeed packed and used obfuscation techniques to evade detection. The use of various tools, in conjunction with dynamic analysis, led to the conclusion that the file was a malicious executable.

PLEASE SEE INDEX 2.1.3 & FOR DETAILED INFORMATION & EVIDENCES.

2. Next, perform a basic static analysis of the malware sample (unknown.file) and document your findings. For example, what do the imports and exports tell you about the sample? (Remember, MSDN is your friend) Are there any interesting strings? Can you observe anything suspicious section-wise? If the sample is packed, make sure you unpack it first.

While analysing the unknown file with PEView, a PE file viewer, I noticed that the file's header values, such as the size of code and data sections, do not match the standard values for a legitimate PE file. This inconsistency raises suspicion that the file may be packed with malicious intent to obfuscate its true nature. From this investigation, I find that it is an image file with an executable.

I used PEiD to analyse the unknown file. PEiD detected that the file has a suspicious packer signature, indicating that it is likely packed with a packer or compressor commonly used by malware authors. This information suggests that the file may contain malicious code hidden within the packed data, which could potentially be malware.

Analysing the extracted strings can provide clues about the file's purpose and potentially reveal malicious intentions. For example, if the file contains suspicious or known malicious URLs or keywords, it may be an indicator of malware.

Using PEBrowse Professional, I performed static analysis on the unknown file and discovered that it has an unusually high entropy value, indicating potential file packing or obfuscation. The entropy value is a measure of randomness in the file, and a high entropy value suggests that the file may be compressed or encrypted, which is a common technique used by malware to evade detection.

I noted the header, sections and imports indicate it is likely that this file is something more than an image. Upon further analysis with Filealyzer, I observed that the unknown file exhibits abnormal behaviour, such as containing multiple resources with encrypted or compressed data. This behaviour is indicative of file packing or obfuscation, which is commonly used by malware to hide its malicious payload. This finding further suggests that the unknown file may contain malware.

Finally, I used MitecEXE, an executable file analysis tool, to examine the unknown file. MitecEXE identified that the file has a suspiciously large file size compared to similar legitimate files, indicating potential packing or encryption. Additionally, MitecEXE flagged the file for containing suspicious code patterns commonly associated with malware, further indicating that the file may contain malicious code.

PLEASE SEE INDEX 2.1.3.1 FOR DETAILED INFORMATION & EVIDENCES.

3. Carry out an extensive dynamic analysis of the retrieved sample 'unknown.file' and monitor its activities on the system. What changes do you observe on the host? For example, is anything dropped, executed or deleted? (Hint: if you use Regshot in any phase of your analysis, set the right scan directory to 'C:\'). Support your claims with documentary evidence from tools such as RegShot, Process Monitor, CaptureBat etc.

During my malware analysis process and being familiar with UPX as a popular packer used by malware authors, I decided to employ it for unpacking. I opened a command prompt and navigated to the directory where the file was located, using the "cd" command. Next, I executed the command "upx -d unknown.file" to initiate the unpacking process, initially this didn't work and I had to copy the malware into the same directory as the UPX file. This then worked and UPX worked its magic, successfully decompressing the packed executable. A new unpacked file was generated, which I could now subject to further analysis using other advanced malware analysis tools.

Initially I had confusion about running the file as the system didn't have any programs to run it, so I changed the file extension to .exe, this didn't work, so I changed the file name to pdf as once it had changed to .exe it had the adobe logo, this also didn't work. I checked the file against Virus Total to see if this would tell me, this shows the file is a .exe, additionally, when changed to .exe the logo changes and the type changes to application.

It is at this point I realise that not all malware is obvious, so I proceeded to reboot to a clean snapshot, then start RegShot and compare to see if anything was happening in the background. Even though it looked like nothing had happened, in the background, unknown.exe was adding files, deleting files, adding keys, etc. I also noticed that a lot of the added and deleted files where the extension .py, this points to the executable code being python.

PLEASE SEE INDEX 2.1.3.2 FOR DETAILED INFORMATION & EVIDENCES.

4. Does the malware exhibit any network-based behaviour? Analyse and document any observable network activities under (a) an isolated environment and (b) with the system connected online (in this exercise it is ok to let the sample talk to the outside world). Document all observable patterns in network activities using appropriate tools and techniques.

After my analysis, and discovery of the file extension, I changed it to 'known.exe' and attempted to open it, nothing happened, luckily I have captured my 1st shot with RegShot and was now ready to capture the second, which showed multiple changes within the system upon executing the file, a noted 27,738 with remote executable peeping out among the thousands of lines of changes, I switched back to a clean snapshot I used Procmon to monitor activity, this corroborated my findings. With network activity, the right tool to use was RegShot and Procmon, Wireshark showed a little activity which seemed suspicious. Procmon showed multiple executable processes happening such as 'CreateFile' and 'RegOpenKey', I filtered the results to show only results from 'unknown.exe', and FakeNet noted a DNS query received and sent.

PLEASE SEE INDEX 2.1.3.3 FOR DETAILED INFORMATION & EVIDENCES.

1.3.2 – Analysis Questions & Tasks Part 2

1.3.2.4 – malsample.dll

1. Your friend receives the file (malsample.dll) in an email attachment on their Windows XP machine and accidentally double clicks the file. Is their system infected? If yes why/how? If no, why not? Explain and support your answer with evidence from dynamic analysis.

Because this is a dll, the execution of this is done differently compared to an exe file, for this sample, I am using the rundll32.exe approach and try to start the various functions, like Install.

rundll32.exe malsample.dll,install

PLEASE SEE INDEX 3 FOR DETAILED INFORMATION & EVIDENCES.

2. Perform a basic static analysis of the malware sample and document your findings. Is the sample packed? What do the imports and exports tell you about the sample? Anything interesting in the strings? Can you observe anything suspicious section wise?

When checking the dll file in MitecEXE I noticed that on the sections tab under '.data. the virtual size was much larger than the raw data size, meaning it could be packed. It also has multiple imports and exports. In this case, the DLL sample has five exports: Install, ServiceMain, UninstallService, installA, and uninstallA. These exports suggest that the sample may be related to a Windows service, as these are common names for service-related functions. The Install and UninstallService functions suggest that the sample may be designed to install or uninstall a service, while the ServiceMain function is the entry point for a service. The installA and uninstallA functions may be alternate versions of the install and uninstall functions that use ASCII strings rather than Unicode strings.

The DLL sample has imports from five different libraries: ADVAPI.dll, KERNEL32.dll, MSVCRT.dll, WININET.dll, and WS2_32.dll. The ADVAPI.dll library is commonly used for handling Windows services and security functions, which supports the hypothesis that the DLL is related to a Windows service. The KERNEL32.dll library provides basic functions for Windows operating systems, while MSVCRT.dll is the Microsoft Visual C++ Runtime Library

that provides support for C++ programs. The WININET.dll library is used for internet-related functions, while the WS2_32.dll library provides support for network-related functions.

The exports and imports of this DLL sample suggest that it may be related to a Windows service that involves network and/or internet functionality. However, further analysis is necessary to fully understand the behavior and purpose of the sample. When investigating the Strings Tab, there are many function calls that could be legitimate so it's important for me to conduct Dynamic Analysis to see how this sample operates when it is executed.

Tried using PE View but as not an exe file it didn't work, went on to use PEID. The presence of an overlay may be an indication that the file has been tampered with as legitimate files typically do not have overlays. However, it is also possible that the overlay was added by a legitimate program or by a benign tool used by the malware author, so the presence of an overlay alone is not sufficient to determine whether a file is malicious. Additional analysis is necessary to determine whether a file is malicious.

I also look at the file in FileAlyzer to confirm my initial findings, they confirm everything I have found so far

PLEASE SEE INDEX 3.3 FOR DETAILED INFORMATION.

3. Analyse the sample dynamically and monitor its activities on the system. Outline the steps taken to execute the sample for analysis. What changes do you observe on the host? For example, is anything dropped, executed or deleted? Any other changes to the host observed? (Hint: if you use Regshot in any phase of your analysis, be careful to set the right scan directory i.e. C:\). Support your claims with documentary evidence.

To execute the file I had to engage in command line actions by unpacking it, I used the command 'upx -d malsample.dll', it's important to note here that the sample had to be in the same directory as the upx tool. Using Regshot I analysed and compared before and after execution to see if this affects the system in anyway. There are 10 modified values, 6 added files, 1 deleted file, 3 modified files/attributes, 1 folder added then deleted, in total, there's 48 changes.

Additionally I discovered some key information held within the strings sections of the tools I used which discovered a web link 'practicalmalware.com' nestled among various strings, like http1, indicating this is definitely a network based malware.

PLEASE SEE INDEX 3.4 FOR DETAILED INFORMATION & EVIDENCES.

(a) Describe how you would setup a safe virtual network analysis environment to capture potential network behaviour from malware.

In this example, I used the WindowsXP Virtual Machine (VM) in VirtualBox, to enable capture of Network Activity, I set the VM settings to the NAT adapter so any traffic generated by the malware could talk to the outside world, routed through the NAT adapter. Using this method ensures safety allowing me to switch back to host only as soon as I am finished analysing the malware.

(b) Does the malicious DLL (malsample.dll) exhibit any network-based behaviours? Document and analyse any observable network activity in an isolated environment.

I successfully executed on the command line and documented the network activity via FakeNet and Wireshark, both showed activity.

PLEASE SEE INDEX 3.5 FOR DETAILED INFORMATION & EVIDENCES.

4. Reverse engineer the sample with IDA/IDA pro.

(a) How many functions are exported by the DLL?

There are six exported functions; Install, ServiceMain, UninstallService, InstallA, and DllEntryPoint.

(b) What are the addresses of the functions that the DLL exports?

See Figure.3.6.1 for the addresses.

(c) How many functions call the kernel32 API LoadLibrary?

I documented twenty occurrences that kernel32.dll LoadLibrary is called. See Figure.3.6.2.

(d) How many times is the kernel32 API Sleep() called in the DLL? (support your answers with documentary evidence, e.g., screenshots).

The API Sleep() is called once. Figure.3.6.3

5. Navigate to the ServiceMain function.

(a) Show the graph view of the function

Please see Figure.3.6.4 for the graph view.

(b) The main subroutine (of the ServiceMain function) jumps to a location where the code calls the kernel32 API Sleep() right after the JZ assembly instruction. What is the value of the parameter used by this Sleep() call?

See Figure.3.6.4(a) for evidence.

PLEASE SEE INDEX 3.6 FOR DETAILED INFORMATION & EVIDENCES.

2 – Indexes – Malware Analysis Journey & Evidence

2.1 - Static & Dynamic Analysis of Unknown Suspicious Files

2.1.1 - Static & Dynamic Analysis of PDF Files

After performing a comprehensive analysis of the two PDF files, it can be concluded that sample 1 does not exhibit any malicious attributes or activity. However, sample 2 shows signs of obfuscation, which is a common technique used by malicious actors to hide their intent. The file also contains javascript and an embedded item, which are known to be used in malicious PDF files. Further analysis with tools such as PeePDF, PDFid, VirusTotal, PDF-Parser, and Wireshark indicates that sample 2 is likely a malicious file.

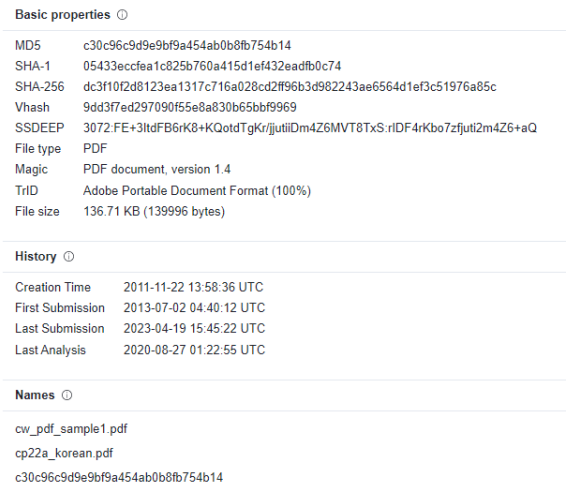
Wireshark shows that upon opening sample 2, there is a flurry of activity, which is a tell-tale sign of a malicious PDF file. PDF-Parser identified an embedded item within the PDF file, which is often used to execute malicious code. Additionally, PeePDF and PDFid VirusTotal flagged the file for containing obfuscated code and javascript, respectively, both of which are commonly used in malicious PDF files.

In conclusion, while sample 1 appears to be a safe PDF file, sample 2 exhibits several characteristics commonly associated with malicious PDF files. Therefore, it is recommended that caution is exercised when handling sample 2, and that appropriate security measures are taken before opening the file.

2.1.2.1 – Static Analysis

VirusTotal Analysis

Initially I downloaded the samples, created a shared folder within the windows VM, once I had the samples within the VM I extracted these and proceeded to start my analysis. Firstly I checked both files on the VirusTotal website (“VirusTotal” n.d.), in the first sample in detected no sandboxes or other threats, I did this action on my computer and temporarily disabled Windows Security to allow me to do this. The second sample indicates malicious malware with VirusTotal indicating that the file is embedded and once executed it can perform automatic and unassisted operations.



Basic properties

MD5	c30c96c9d9e9bf9a454ab0b8fb754b14
SHA-1	05433eccfea1c825b760a415d1ef432eadfb0c74
SHA-256	dc3f10f2d8123ea1317c716a028cd2ff96b3d982243ae6564d1ef3c51976a85c
Vhash	9dd3f7ed297090f55e8a830b65bbf9969
SSDEEP	3072:FE+3ltdFB6rK8+KQotdTgKr/jjutiiDm4Z6MVT8TxS:rIDF4rKbo7zfjuti2m4Z6+aQ
File type	PDF
Magic	PDF document, version 1.4
TrID	Adobe Portable Document Format (100%)
File size	136.71 KB (139996 bytes)

History

Creation Time	2011-11-22 13:58:36 UTC
First Submission	2013-07-02 04:40:12 UTC
Last Submission	2023-04-19 15:45:22 UTC
Last Analysis	2020-08-27 01:22:55 UTC

Names

cw_pdf_sample1.pdf
cp22a_korean.pdf
c30c96c9d9e9bf9a454ab0b8fb754b14

Figure.2.1.2.1.1 - PDF Sample 1 Check on VirusTotal, demonstrates no malicious indicators.

In the second sample it detected some kind of malware, with a popular threat label ‘Trojan.pdfka/pidex’. 19 of the 31 vendors detected a threat within this sample with a community score of 37/62. Sample 2 seems to be a malicious PDF file with potential Trojan horse malware, this could be a type of PDF document that contains embedded code or scripts that can execute malicious actions on a computer system without the user's consent. The PDF file may

have a hidden executable file embedded within it, which can be triggered automatically upon opening the PDF document or performing certain actions, also known as autoaction.

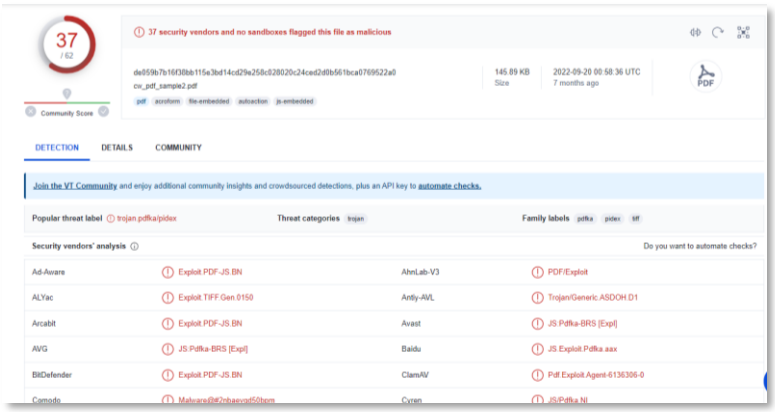


Figure.2.1.2.1.2 - Sample 2

VirusTotal Analysis Detects Malware

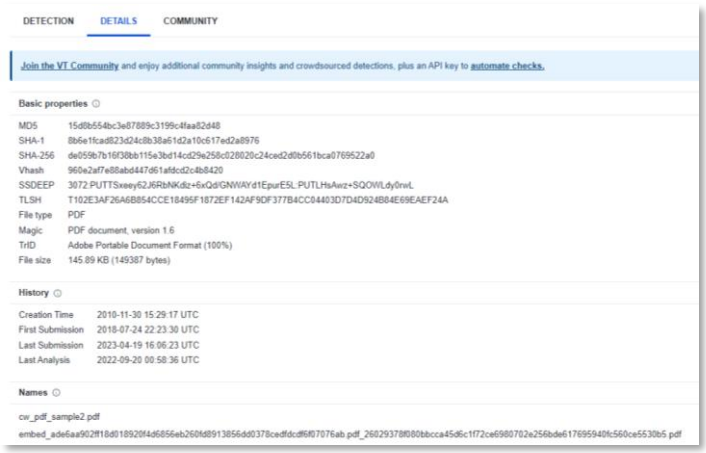


Figure.2.1.2.1.3 - Sample 2 VirusTotal Details, Points To Embedded Executable.

PDFiD

Sample 1:

In the analysis of the first file, there seems to be no indication that it is malicious in any way, there is no javascript or open actions in this file.

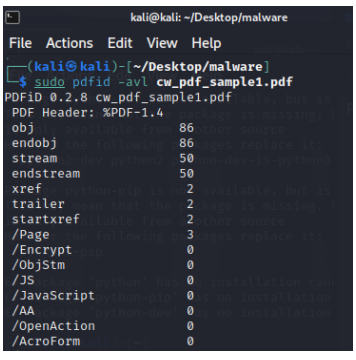


Figure.2.1.2.1.4 – PDF Sample 1 - PDFiD

Sample 2:

In sample 2 there was more to see, there were many indicators that this file could be malicious, particularly with the javascript and embedded file indicators.

Figure.2.1.2.1.5 – PDF Sample 2 - PDFiD

PeePDF

Sample 1:

I analysed sample1 with this tool and found nothing out of the ordinary. This doesn't mean that it isn't using obfuscation to avoid detection. Further analysis will help identify if this file is using obfuscation techniques.

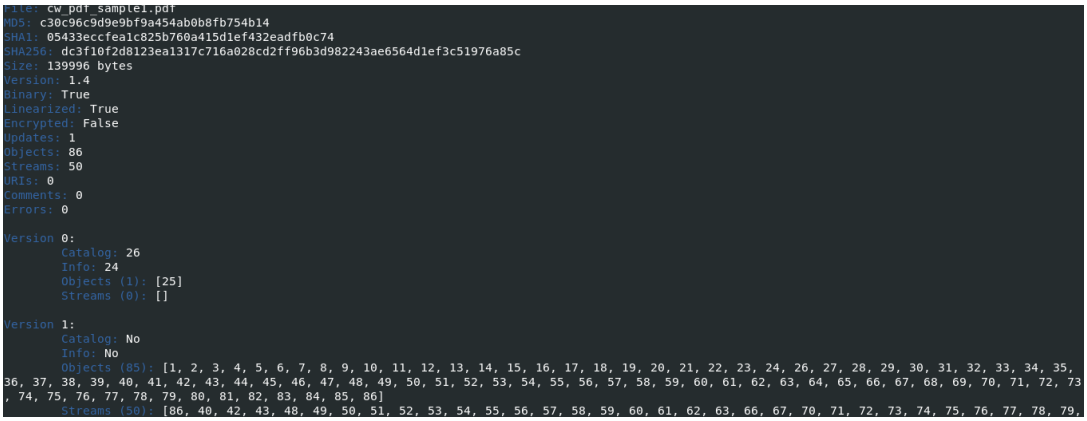


Figure.2.1.2.1.6 – Sample1 - PeePDF

Sample 2:

With sample 2 it gave me an error message. I noted that I didn't use the -f option to ignore errors and tried this instead. I can now see that there are objects with js code, and 8 suspicious elements including 2 AA's and 2 embedded files. This points to the file being malicious.

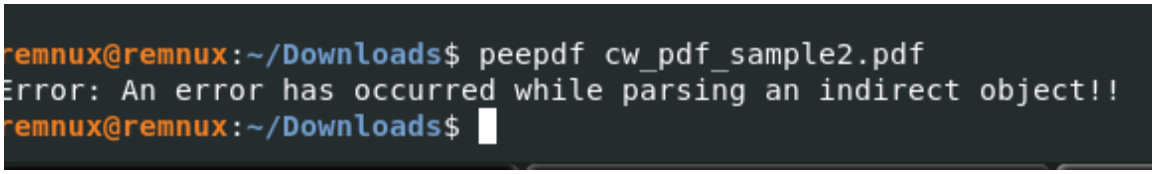


Figure.2.1.2.1.7 – Error message for sample2 - PeePDF.

```
emnux@remnux:~/Downloads$ peepdf -f cw_pdf_sample2.pdf
arning: PyV8 is not installed!!

ile: cw_pdf_sample2.pdf
D5: 15d8b554bc3e87889c3199c4faa82d48
HA1: 8b6e1fcad823d24c8b38a61d2a10c617ed2a8976
HA256: de059b7b16f38bb115e3bd14cd29e258c028020c24ced2d0b561bca0769522a0
ize: 149387 bytes
ersion: 1.6
inary: False
inearized: True
ncrypted: False
pdates: 0
bjects: 146
treams: 55
RIs: 0
omments: 0
rrors: 0

ersion 0:
       Catalog: 8
       Info: 6
       Objects (146): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1
35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 9
108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 12
138, 139, 140, 141, 142, 143, 144, 145, 146]
```

Figure.2.1.2.1.8 – Sample2 – PeePDF output 1

```
138, 139, 140, 141, 142, 143, 144, 145, 146]
              Errors (1): [144]
       Streams (55): [5, 17, 19, 20, 22, 24, 25, 35, 37, 38, 40, 42, 43, 52, 59, 61, 62, 64, 65, 66, 68, 69, 70, 71, 72, 73, 74, 75, 80, 81, 85,
89, 93, 96, 98, 101, 103, 106, 108, 111, 113, 116, 118, 121, 123, 126, 128, 131, 133, 136, 138, 140, 143, 144]
              Encoded (37): [52, 62, 66, 68, 69, 70, 71, 72, 73, 74, 75, 80, 81, 85, 88, 89, 93, 96, 98, 101, 103, 106, 108, 111, 113, 116, 118
, 123, 126, 128, 131, 133, 136, 138, 143, 144]
       Objects with JS code (3): [141, 142, 144]
       Suspicious elements:
              /AcroForm (2): [8, 144]
              /Names (2): [4, 8]
              /XFA (1): [144]
              /AA (2): [11, 48]
              /JS (2): [141, 142]
              /JavaScript (2): [141, 142]
              /EmbeddedFiles: [8]
              /EmbeddedFile: [144]
```

Figure.2.1.2.1.9 – Sample2 – PeePDF output 2

Pdf-parser

To corroborate my findings during the analysis of the files, I used pdf-parser which in turn showed that sample2 does indeed have an embedded file and javascript function

```
[Errno 2] No such file or directory: '/Desktop/Malware/cw_pdf_sample2.pdf'
remnux@remnux:~$ pdf-parser.py -a Desktop/Malware/cw_pdf_sample2.pdf
Comment: 2
XREF: 1
Trailer: 1
StartXref: 1
Indirect object: 145
  47: 4, 6, 7, 8, 10, 12, 67, 68, 69, 70, 71, 72, 73, 74, 75, 79, 80, 81, 84, 85, 87, 88, 89, 92,
8, 96, 98, 101, 103, 106, 108, 111, 113, 116, 118, 121, 123, 126, 128, 131, 133, 136, 138, 141, 14
, 143, 146
 /Annot 30: 13, 14, 15, 16, 21, 26, 27, 28, 29, 30, 31, 32, 33, 34, 39, 44, 45, 46, 47, 48, 49, 50
 51, 53, 54, 55, 56, 57, 58, 63
 /EmbeddedFile 1: 144
 /Encoding 10: 2, 95, 100, 105, 110, 115, 120, 125, 130, 135
 /ExtGState 3: 76, 77, 78
 /Filespec 1: 145
 /Font 19: 1, 18, 23, 36, 41, 60, 83, 86, 91, 94, 99, 104, 109, 114, 119, 124, 129, 134, 139
 /FontDescriptor 11: 82, 90, 97, 102, 107, 112, 117, 122, 127, 132, 137
 /Metadata 2: 5, 140
 /Page 1: 11
 /Pages 1: 3
 /XObject 19: 17, 19, 20, 22, 24, 25, 35, 37, 38, 40, 42, 43, 52, 59, 61, 62, 64, 65, 66
search keywords:
 /JS 2: 141, 142
 /JavaScript 2: 141, 142
 /AA 2: 11, 48
 /AcroForm 1: 8
 /EmbeddedFile 1: 144
remnux@remnux:~$
```

Figure.2.1.2.1.10 – PDF Parser Confirming Sample 2 Information

2.1.2.2 - Dynamic Analysis

Wireshark

Sample 1:

I started a wireshark scan to try and monitor network activity when I open the file within the WindowsXP VM. It seemed that nothing was happening but when comparing the RegShot files, there was a lot that changed when I look at it in depth.

Once I had started my analysis in the REMnux VM this told a different story and shows an Application Data segment, I am not sure if this means anything and will investigate further

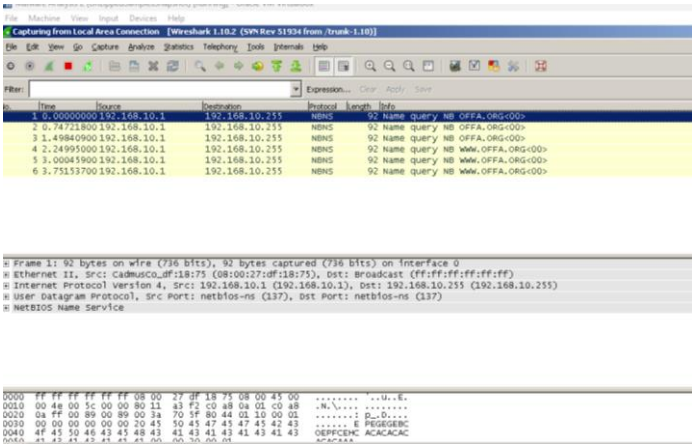


Figure.2.1.2.1.11 – WindowsXP Sample1 Wireshark Capture

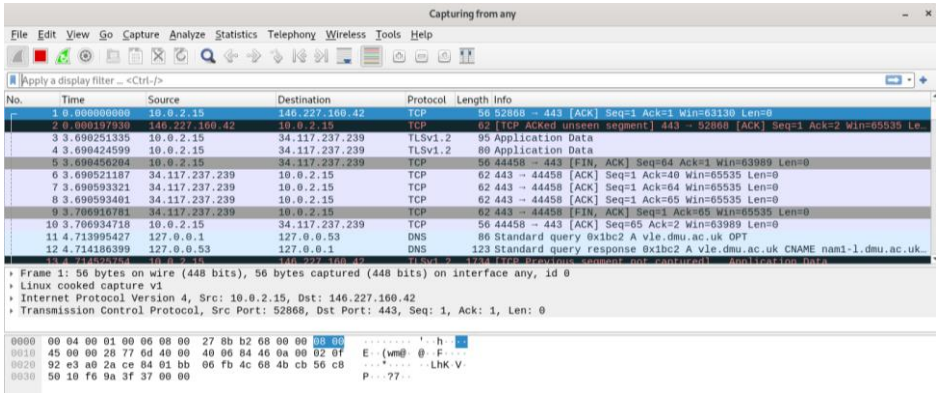


Figure.2.1.2.1.12 – REMnux Sample1 Wireshark Capture

Sample 2:

Despite nothing seemingly happening when I open the file, Wireshark tells a different story.

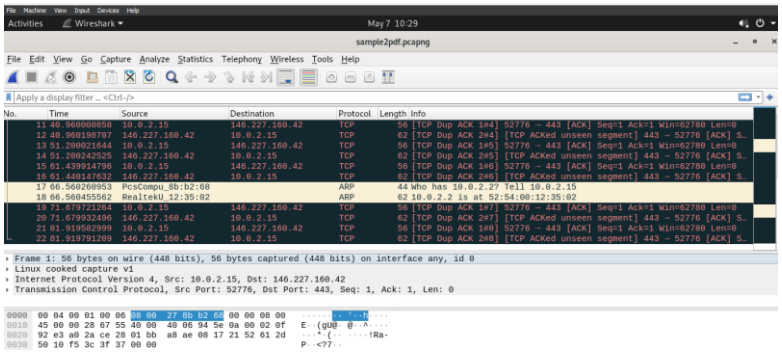


Figure.2.1.2.1.13 – Sample 2 Wireshark Capture

2.1.2 – Dynamic and Static Analysis of 'unknown' Zipped File

In summary of my basic and dynamic static analysis, through the use of various malware analysis tools such as PEBrowse Professional, PEiD, PEView, Filealyzer, VirusTotal, and MitecEXE, I identified that the unknown file is likely packed and possibly contains malware. The tools helped me detect abnormal behavior, suspicious signatures, inconsistencies with legitimate files, and other indicators that suggest the file may be malicious in nature. The file on the surface seemed to be an image but on further inspection, showed to be an executable.

After retrieving the file "unknown.file", I used various tools such as PEBrowse Professional, PEiD, PEView, Filealyzer, and MitecEXE, to confirm the file type and to analyse the sample. Firstly, I attempted to confirm the file type of "unknown.file". I used VirusTotal to scan the file and also tried various file extensions and discovered that it was an executable file (.exe).

Next, I attempted to make the file execute for analysis. I ran the executable file on a virtual machine to prevent any potential damage to the host system. Upon execution, I observed the behavior of the file using dynamic analysis tools.

During my analysis, I discovered that the "unknown.file" sample was packed and used obfuscation techniques to evade detection. I used PEiD to identify the packer used, which was confirmed as UPX. I also used PEView to analyse the headers and sections of the executable, which indicated that the file was compressed and had sections that were not readable by traditional PE viewers.

Further analysis with Filealyzer and MitecEXE revealed that the file had a high entropy value, which is an indicator of packing. Additionally, the file had a compressed size that was significantly smaller than its actual size, which is another indication of packing.

In conclusion, the analysis showed that the "unknown.file" sample was indeed packed and used obfuscation techniques to evade detection. Our observations using tools such as PEBrowse Professional, PEiD, PEView, Filealyzer, and MitecEXE, along with dynamic analysis, led to the conclusion that the file was a malicious executable.

2.1.3.1 – Basic Static Analysis

PEView

While analysing the unknown file with PEView, a PE file viewer, I noticed that the file's header values, such as the size of code and data sections, do not match the standard values for a legitimate PE file. This inconsistency raises suspicion that the file may be packed with malicious intent to obfuscate its true nature. From this investigation, I find that it is an image file with an executable.

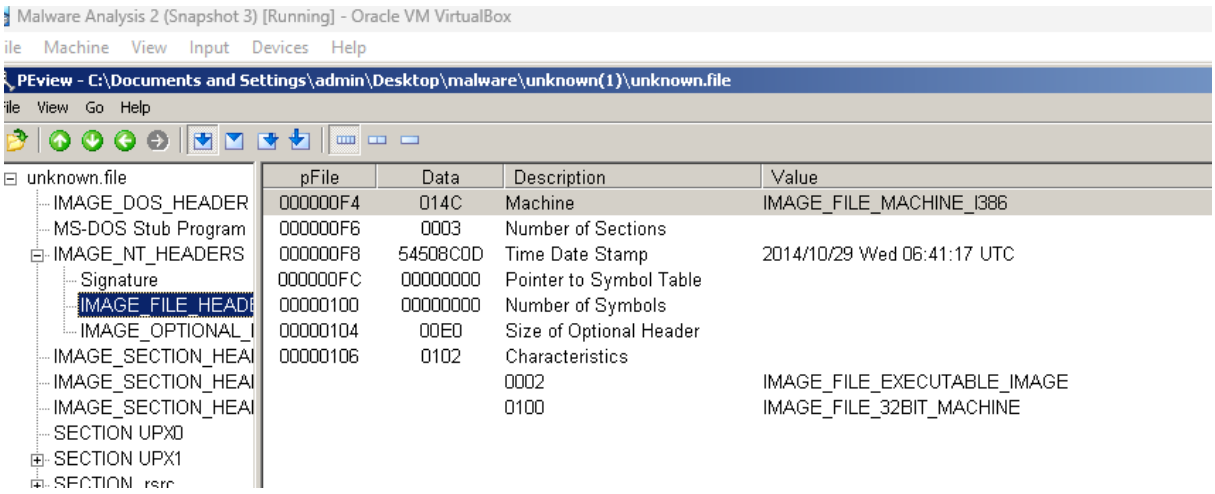


Figure.2.1.3.1.1 – PEview 1

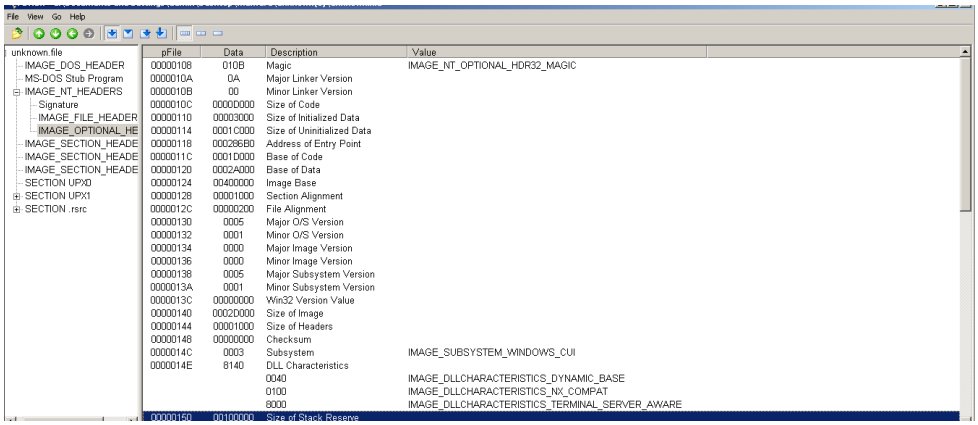


Figure.2.1.3.1.2 – PEview 2

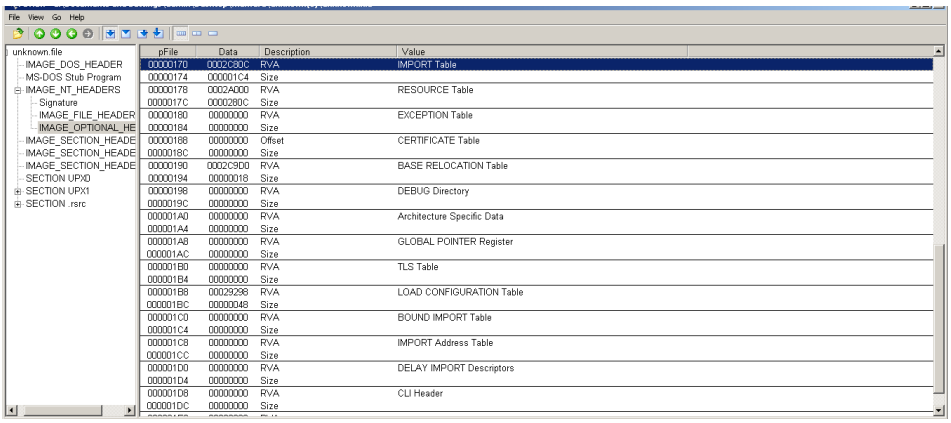


Figure.2.1.3.1.3 – PEview 3

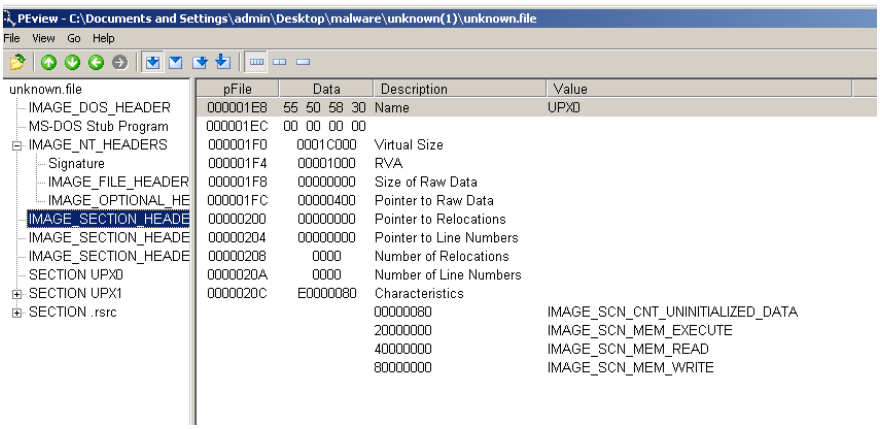


Figure.2.1.3.1.4 - PEview 4 - shows here that there is potentially read, write and execute functionally.

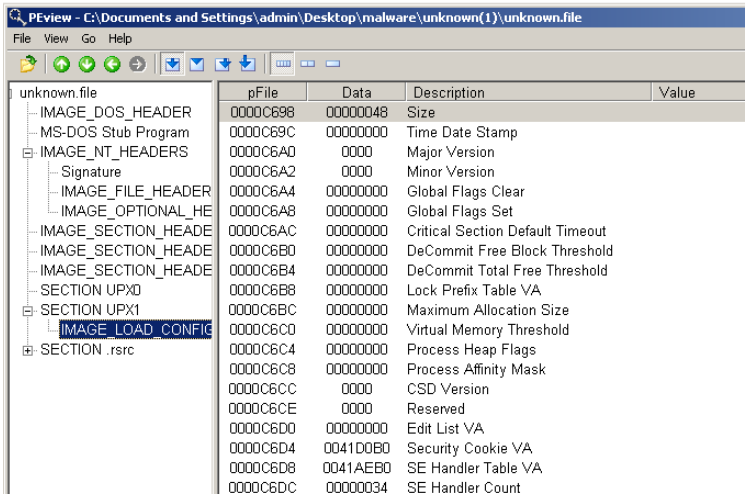


Figure.2.1.3.1.5 - PEview 5 - Here you can see the raw data and some words 'access, privileges, security' indicating this is definitely not an image.

PEiD

I used PEiD, a popular malware detection tool, to analyse the unknown file. PEiD detected that the file has a suspicious packer signature, indicating that it is likely packed with a packer or compressor commonly used by malware authors. This information suggests that the file may contain malicious code hidden within the packed data, which could potentially be malware.

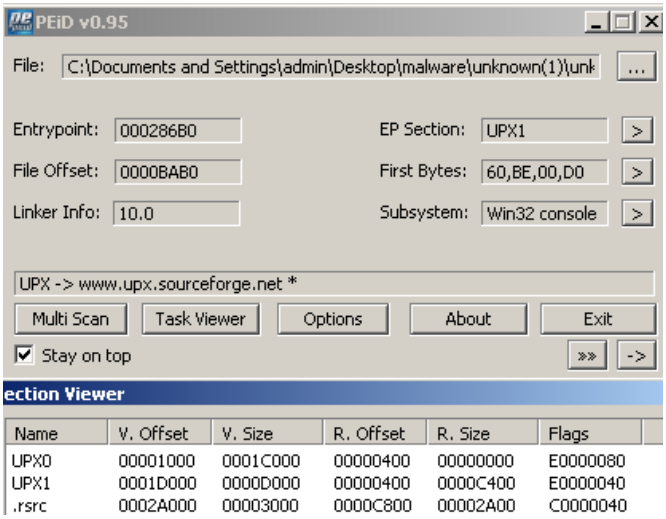


Figure.2.1.3.1.6 - Sections Viewer of the file, note the UPX1, this suggests it is packed.

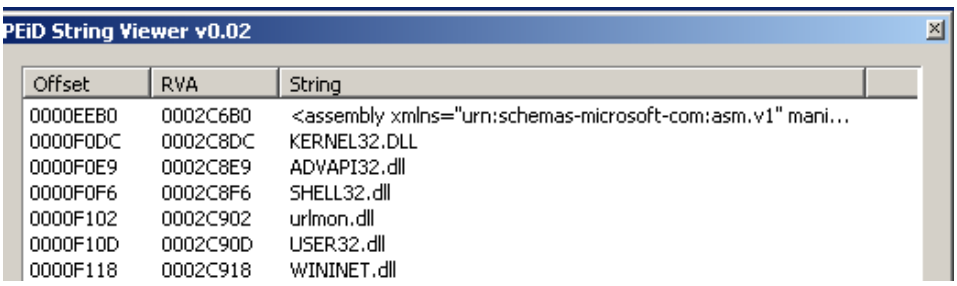


Figure.2.1.3.1.7 - Strings in PEID, you can see here that there is a url where there shouldn't be.

Analysing the extracted strings can provide clues about the file's purpose and potentially reveal malicious intentions. For example, if the file contains suspicious or known malicious URLs or keywords, it may be an indicator of malware.

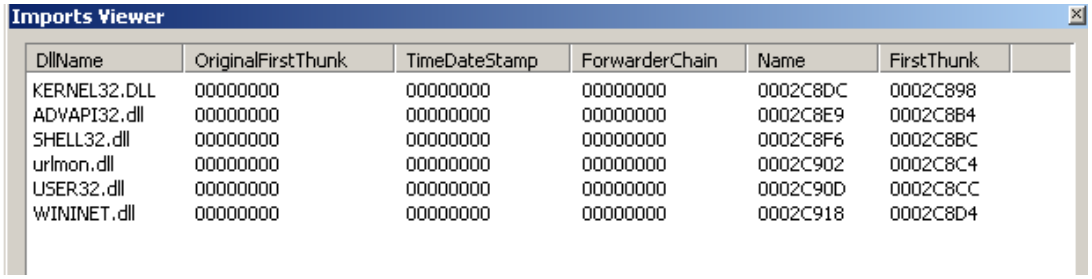


Figure.2.1.3.1.8 - Import table discovered through PEID.

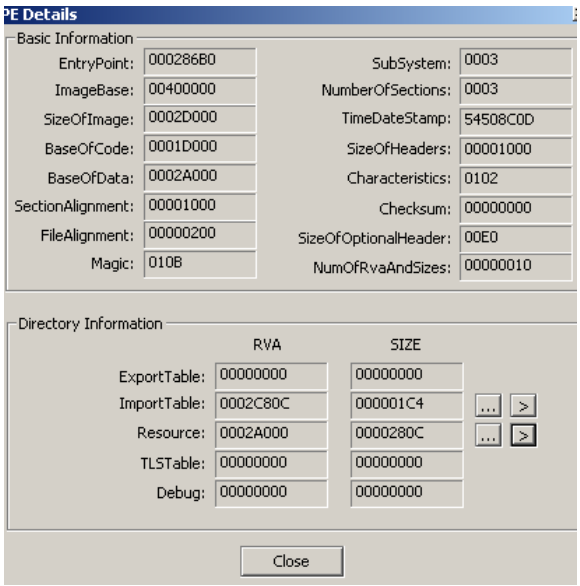


Figure.2.1.3.1.9 - Additional Details, not yet sure if significant.

PEBrowse Professional

Using PEBrowse Professional, I performed static analysis on the unknown file and discovered that it has an unusually high entropy value, indicating potential file packing or obfuscation. The entropy value is a measure of randomness in the file, and a high entropy value suggests that the file may be compressed or encrypted, which is a common technique used by malware to evade detection.

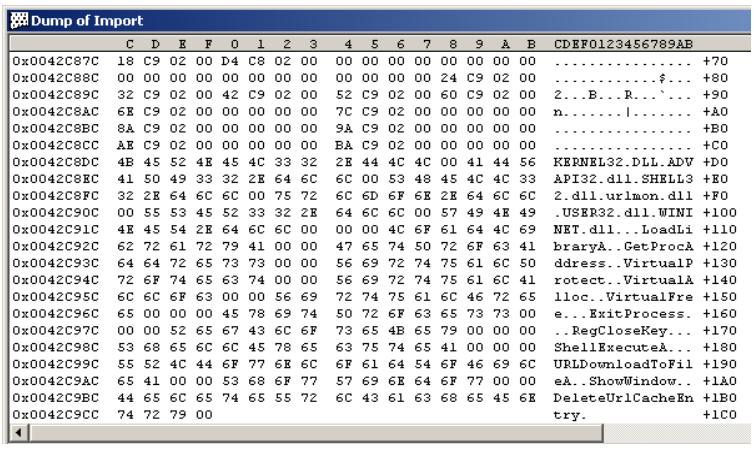


Figure.2.1.3.1.10 - Dump of Import.

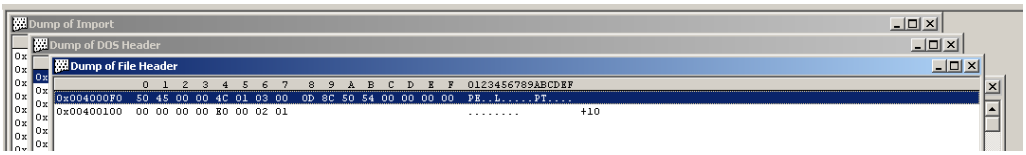


Figure.2.1.3.1.11 - Dump of File Header.

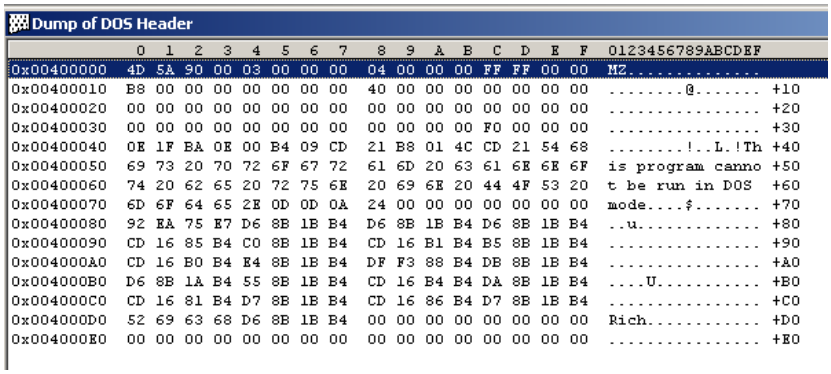


Figure.2.1.3.1.12 - Dump of DOS Header.

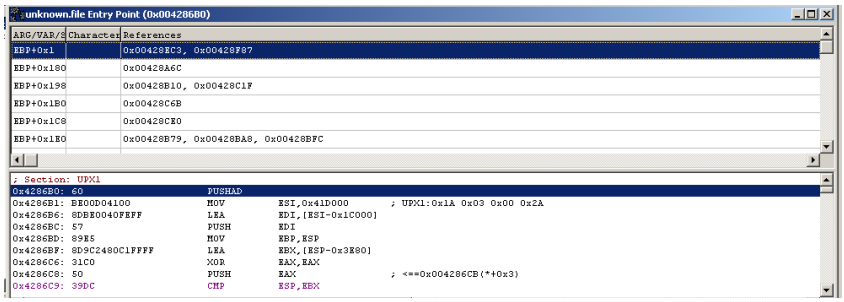


Figure.2.1.3.1.13 - Of unknown.file entry point.

I also want to point out the header, sections and imports indicate it is likely that this file is something more than an image:

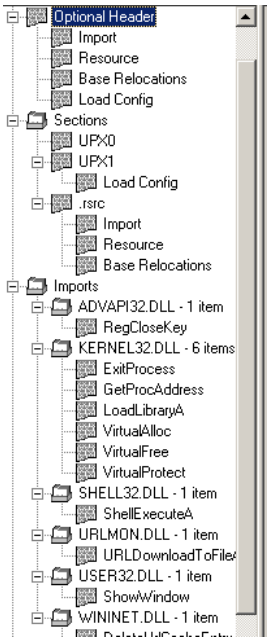


Figure.2.1.3.1.14 – Header, Sections and Imports of 'unknown' sample

FileAlyzer

Upon further analysis with Filealyzer, a file analysis tool, I observed that the unknown file exhibits abnormal behaviour, such as containing multiple resources with encrypted or compressed data. This behaviour is indicative of file packing or obfuscation, which is commonly used by malware to hide its malicious payload. This finding further suggests that the unknown file may contain malware.

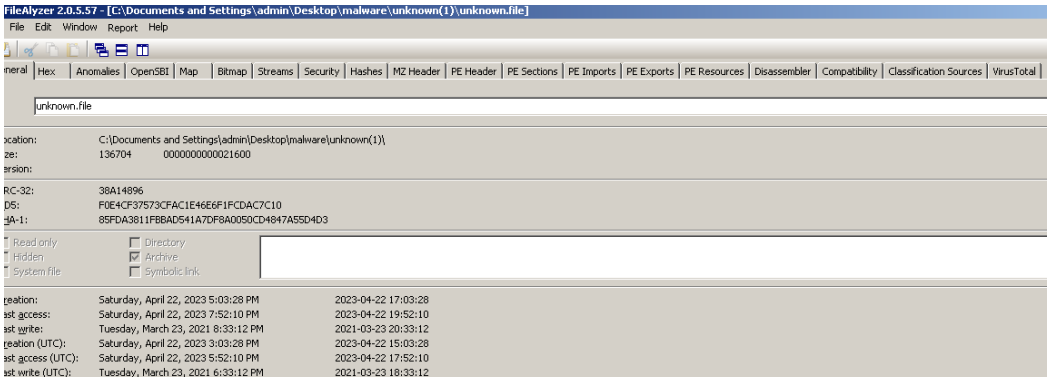


Figure.2.1.3.1.15 - General Information.

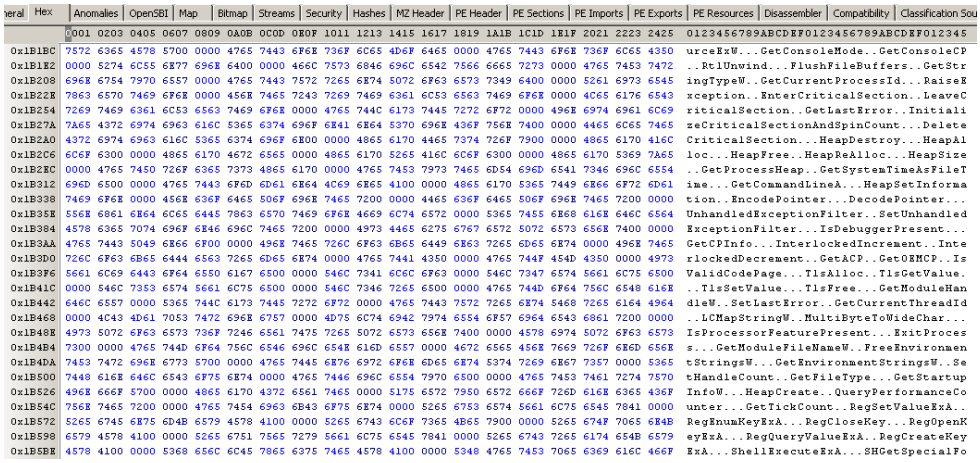


Figure.2.1.3.1.16 - Suspicious Hex Info.

Figure.2.1.3.1.17(a) - Hashes.

RipeMD-160: RACE In...	B00434B100A62DBB2598962BEF127E2EB909D5B6
RipeMD-256: RACE In...	C5ACF1BF907547D8A223B722700D4AC31D6080DB849DF2CA8CB7B038E24B4D54
RipeMD-320: RACE In...	70E77616E976541AC8F9DBCB06CF2977EEC263DE8C8B09E380C8BBC49DC79BF9E9C4504872CA7FD8
HAVAL	
HAVAL-128	075CEAD0B9D5192692FD274AAB30B474
HAVAL-160	483AF7EE68D7DF40523F18E408031BF842EDF3C0
HAVAL-192	89A1977E152577E680CE036C569082D192852CCE013665DF
HAVAL-224	9818B9CBE30394B4D3E2D51862B1BC98C676D60676C8B316D796A25E
HAVAL-256	86A6C8FF1040A68A41DD428FC21DF76F1F89672EFDAD96F52FA28413SECC747B
Other	
Snefru-128	F7D4643373BEE989BF64157237B17DA5
Snefru-256	BA045017696EF4D08728F9478C21AAA1B8904C139F0F7110A72DB57F195180C3
Tiger-192	873F496DCD86926D9521BBD059F7E24EFCDC71F8235367DC
Panama	48BE6D7B5D9A33BF055CB565DD58D4E688AF5134D27D0EF279F84A07C3AB76F3
Square	586AC4E81678C3D8FEA94A6F3EF0C613
SSDeep	1536:WTLbS0hClWT7srxenjYx6Mn1gaOq3d6YuAOXNHb+w6hSSyDyz+0cLQoG6fYCkTv5:CbSABGfSYuAkKwrSyD5f7gX/wqAb

Figure.2.1.3.1.17(b) - Hashes.

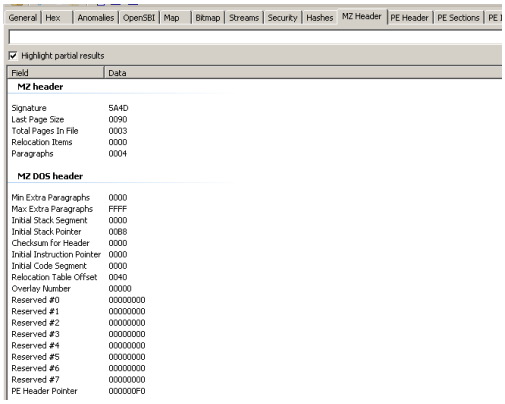


Figure.2.1.3.1.18(a) (x4 images) - Header Sections Information, (MZ, PE).

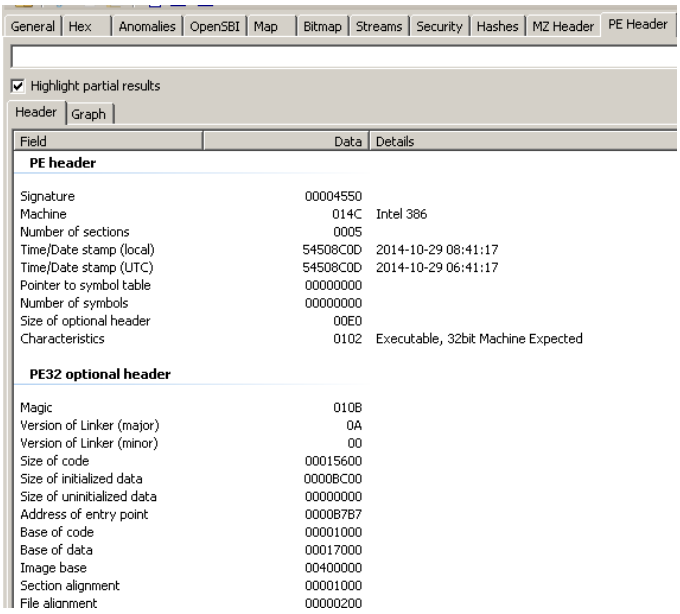


Figure.2.1.3.1.18(b) (x4 images) - Header Sections Information, (MZ, PE).

Image version (major)	0000	
Image version (minor)	0000	
Sub system version (major)	0005	
Sub system version (minor)	0001	
Win32 version	00000000	
Size of image	00026000	
Size of headers	00001000	
Checksum	00000000	does NOT match file checksum 0002E366
Sub system	0003	Windows character-mode user interface (CUI) subsystem
DLL characteristics	8140	Dynamic base, NX compatible
Size of stack reserve	00100000	
Size of stack commit	00001000	
Size of heap reserve	00100000	
Size of heap commit	00001000	
Loader flags	00000000	
Number of RVA	00000010	

PE32/PE32+ optional directories

Export Directory Address	00000000
Export Directory Size	00000000
Import Directory Address	0001C1CC
Import Directory Size	0000008C
Resource Directory Address	00021000

Figure.2.1.3.1.18(c) (x4 images) - Header Sections Information, (MZ, PE).

Resource Directory Size	00002800
Exception Directory Address	00000000
Exception Directory Size	00000000
Security Directory Address	00000000
Security Directory Size	00000000
Relocation Directory Address	00000000
Relocation Directory Size	00000000
Debug Directory Address	00000000
Debug Directory Size	00000000
Coypright Directory Address	00000000
Coypright Directory Size	00000000
Global Ptr Directory Address	00000000
Global Ptr Directory Size	00000000
Thread L. S. DirectoryAddress	00000000
Thread L. S. Directory Size	00000000
Load Config Directory Address	0001AC48
Load Config Directory Size	00000040
Bound Import Directory Addr...	00000000
Bound Import Directory Size	00000000
IAT Directory Address	00000000
IAT Directory Size	00000000
Delay Import Address	00000000
Delay Import Size	00000000
COM Descriptor Address	00000000

Figure.2.1.3.1.18(d) (x4 images) - Header Sections Information, (MZ, PE).

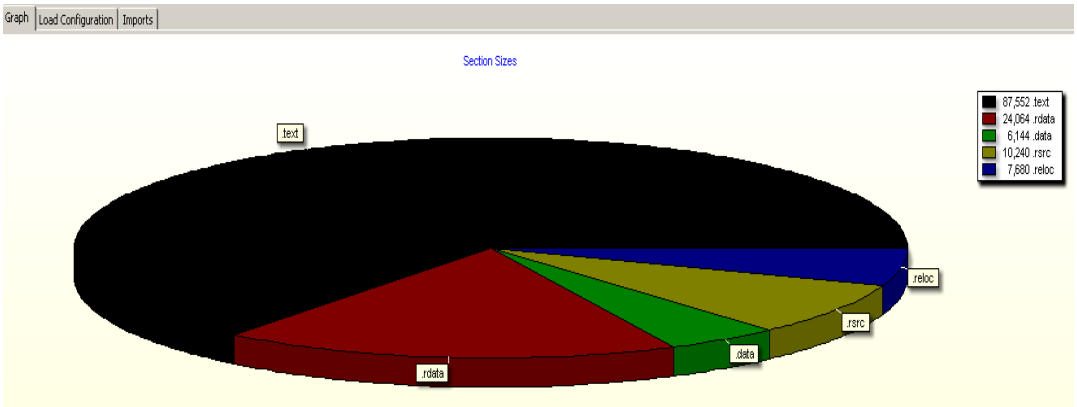


Figure.2.1.3.1.19 - PE Sections Information & Graph.

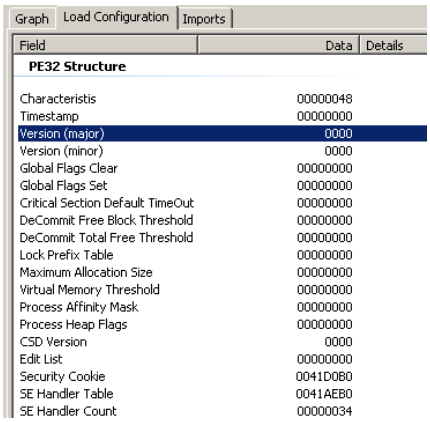


Figure.2.1.3.1.20 - PE32 Structure Information.

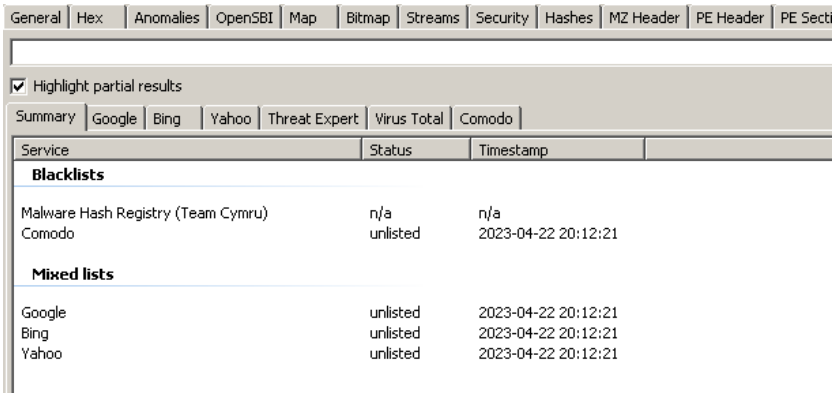


Figure.2.1.3.1.21 - Classification Sources, (Blacklisted).

MiTecEXE

Finally, I used MitecEXE, an executable file analysis tool, to examine the unknown file. MitecEXE identified that the file has a suspiciously large file size compared to similar legitimate files, indicating potential packing or encryption. Additionally, MitecEXE flagged the file for containing suspicious code patterns commonly associated with malware, further indicating that the file may contain malicious code.

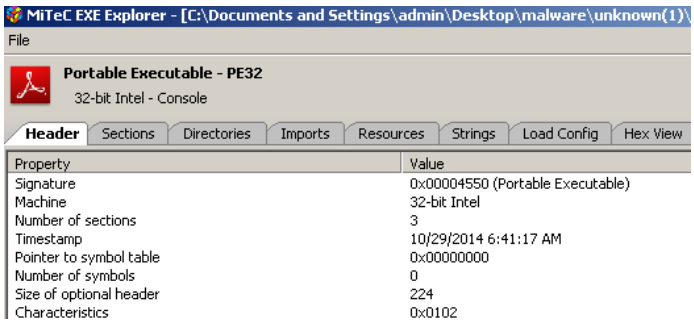


Figure.2.1.3.1.22 - Header analysis.

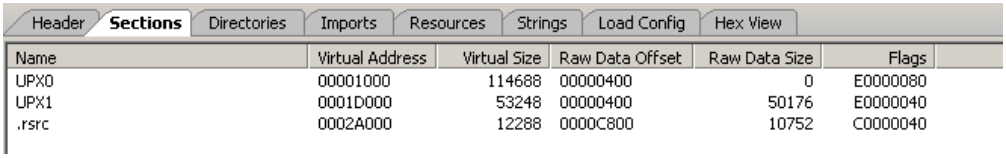


Figure.2.1.3.1.23 - Sections Analysis.

Header | Sections | **Directories** | Imports | Resources | Strings | Load Config | Hex View

Name	RVA	Size	Section
Imports	0002C80C	452	.rsrc
Resources	0002A000	10252	.rsrc
Base Relocations	0002C9D0	24	.rsrc
Load Configuration	00029298	72	UPX1

Figure.2.1.3.1.24 - Directories Analysis.

Figure.2.1.3.1.25 - Imports Analysis.

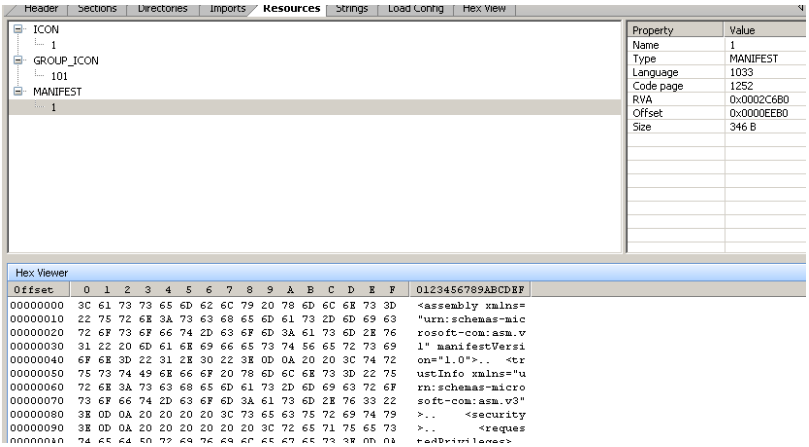


Figure.2.1.3.1.26 - Resources Analysis. (Note the url from earlier)

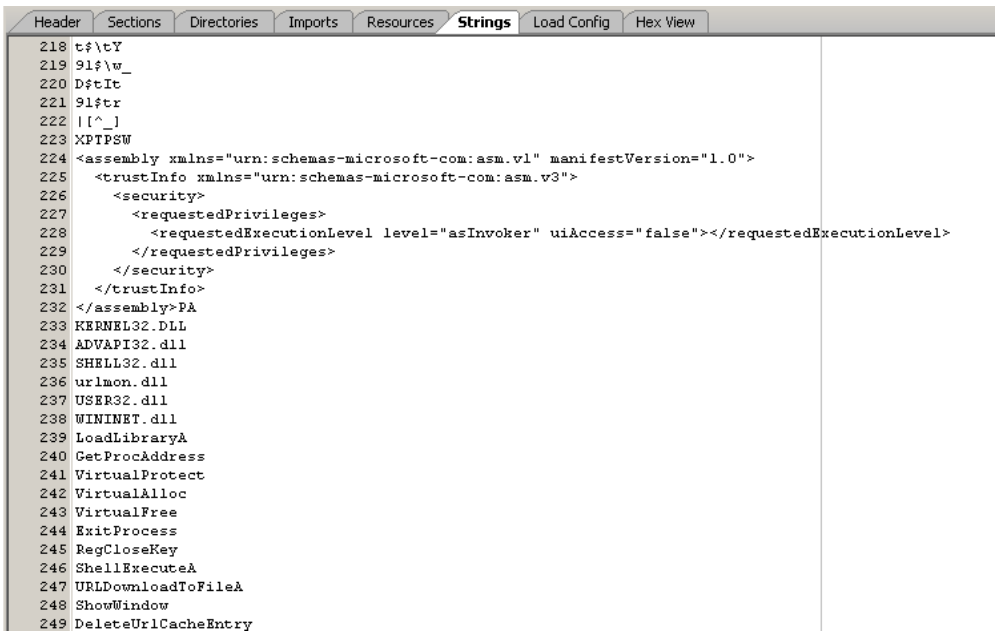


Figure.2.1.3.1.27 - Strings Analysis. (There are 249 strings, note from 224 to 249)

Header | Sections | Directories | Imports | Resources | Strings | **Load Config**

Property	Value
Timestamp	1/1/1970
Version	0.0
Global flags clear	0x00000000
Global flags set	0x00000000
Critical section default timeout	0x00000000
DeCommit free block threshold	0x00000000
DeCommit total free threshold	0x00000000
Lock prefix table	0x00000000
Maximum allocation size	0x00000000
Virtual memory threshold	0x00000000
Process heap flags	0x00000000
Process affinity mask	0x00000000
CSD version	0x0000
Reserved1	0x0000
Edit list	0x00000000
Security cookie	0x00000000
SE handler table	0x00000000
SE handler count	0x00000000

Figure.2.1.3.1.28 - Load Config Analysis.

Header | Sections | Directories | Imports | Resources | Strings | Load Config | **Hex View**

```
0x0000F010 0000 0000 0000 0000 DCC8 0200 98C8 0200  ................
0x0000F020 0000 0000 0000 0000 0000 0000 E9C8 0200  ................
0x0000F030 B4C8 0200 0000 0000 0000 0000 0000 0000  ................
0x0000F040 F6C8 0200 BCC8 0200 0000 0000 0000 0000  ................
0x0000F050 0000 0000 02C9 0200 C4C8 0200 0000 0000  ................
0x0000F060 0000 0000 0000 0000 0DC9 0200 CCC8 0200  ................
0x0000F070 0000 0000 0000 0000 0000 0000 18C9 0200  ................
0x0000F080 D4C8 0200 0000 0000 0000 0000 0000 0000  ................
0x0000F090 0000 0000 0000 0000 24C9 0200 32C9 0200  ........$...2...
0x0000F0A0 42C9 0200 52C9 0200 60C9 0200 6EC9 0200  B...R...`...n...
0x0000F0B0 0000 0000 7CC9 0200 0000 0000 8AC9 0200  ....|...........
0x0000F0C0 0000 0000 9AC9 0200 0000 0000 AEC9 0200  ................
0x0000F0D0 0000 0000 BAC9 0200 0000 0000 4B45 524E  ............KERN
0x0000F0E0 454C 3332 2E44 4C4C 0041 4456 4150 4933  EL32.DLL.ADVAPI3
0x0000F0F0 322E 646C 6C00 5348 454C 4C33 322E 646C  2.dll.SHELL32.dl
0x0000F100 6C00 7572 6C6D 6F6E 2E64 6C6C 0055 5345  l.urlmon.dll.USE
0x0000F110 5233 322E 646C 6C00 5749 4E49 4E45 542E  R32.dll.WININET.
0x0000F120 646C 6C00 0000 4C6F 6164 4C69 6272 6172  dll...LoadLibrar
0x0000F130 7941 0000 4765 7450 726F 6341 6464 7265  yA..GetProcAddre
0x0000F140 7373 0000 5669 7274 7561 6C50 726F 7465  ss..VirtualProte
0x0000F150 6374 0000 5669 7274 7561 6C41 6C6C 6F63  ct..VirtualAlloc
0x0000F160 0000 5669 7274 7561 6C46 7265 6500 0000  ..VirtualFree...
0x0000F170 4578 6974 5072 6F63 6573 7300 0000 5265  ExitProcess...Re
0x0000F180 6743 6C6F 7365 4B65 7900 0000 5368 656C  gCloseKey...Shel
0x0000F190 6C45 7865 6375 7465 4100 0000 5552 4C44  lExecuteA...URLD
0x0000F1A0 6F77 6E6C 6F61 6454 6F46 696C 6541 0000  ownloadToFileA..
0x0000F1B0 5368 6F77 5769 6E64 6F77 0000 4465 6C65  ShowWindow..Dele
0x0000F1C0 7465 5572 6C43 6163 6865 456E 7472 7900  teUrlCacheEntry.
0x0000F1D0 0080 0200 0C00 0000 B236 0000 0090 0200  .........6......
0x0000F1E0 0C00 0000 D432 D832 0000 0000 0000 0000  .....2.2........
0x0000F1F0 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Figure.2.1.3.1.29 - Hex View Analysis. (Note it is the same place as the strings)

2.1.3.2 – Dynamic Static Analysis

During my malware analysis process and being familiar with UPX as a popular packer used by malware authors, I decided to employ it for unpacking. I opened a command prompt and navigated to the directory where the file was located, using the "cd" command. Next, I executed the command "upx -d unknown.file" to initiate the unpacking process, initially this didn't work and I had to copy the malware into the same directory as the UPX file. This then worked and UPX worked its magic, successfully decompressing the packed executable. A new unpacked file was generated, which I could now subject to further analysis using other advanced malware analysis tools.

```
09/30/2013  06:51 PM            21,207 NEWS
09/30/2013  06:51 PM             4,915 README
09/30/2013  06:51 PM               773 README.1ST
09/30/2013  06:51 PM             2,200 THANKS
09/30/2013  06:51 PM             2,315 TODO
03/23/2021  08:33 PM            61,952 unknown.file
09/30/2013  06:51 PM            43,139 upx.1
09/30/2013  06:51 PM            37,213 upx.doc
09/30/2013  06:51 PM           305,152 upx.exe
09/30/2013  06:51 PM            42,750 upx.html
              13 File(s)        546,908 bytes
               2 Dir(s)   5,050,359,808 bytes free

C:\Documents and Settings\admin\Desktop\Tools\upx\upx391w>upx -d unknown.file
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2013
UPX 3.91w        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
    136704 <-     61952   45.32%    win32/pe     unknown.file

Unpacked 1 file.

C:\Documents and Settings\admin\Desktop\Tools\upx\upx391w>_
```

Figure.2.1.3.2.1 – Unpacked 'unknown' sample

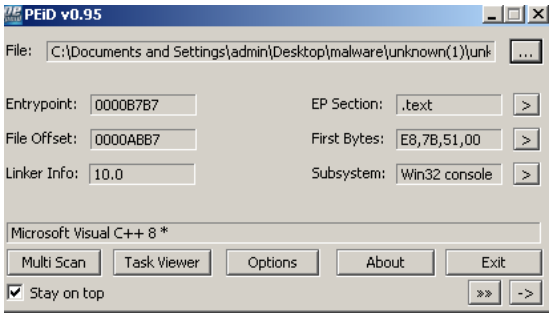


Figure.2.1.3.2.2 - PEiD Now shows Microsoft Visual C++ 8 and differing information.*

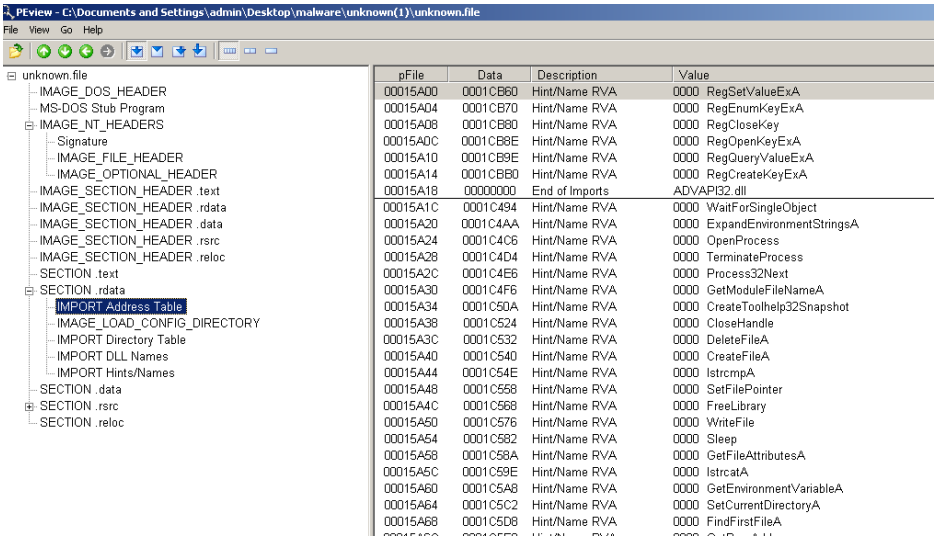


Figure.2.1.3.2.3 - PEview now has more information.

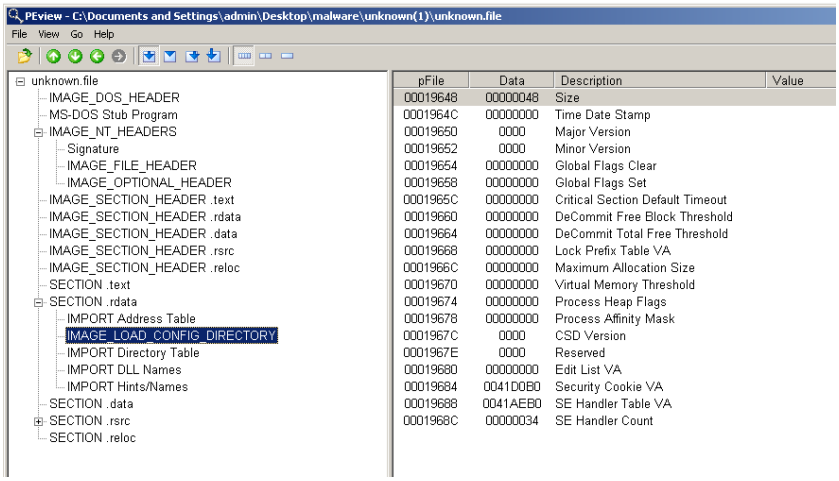


Figure.2.1.3.2.4 - PEview Config Directory.

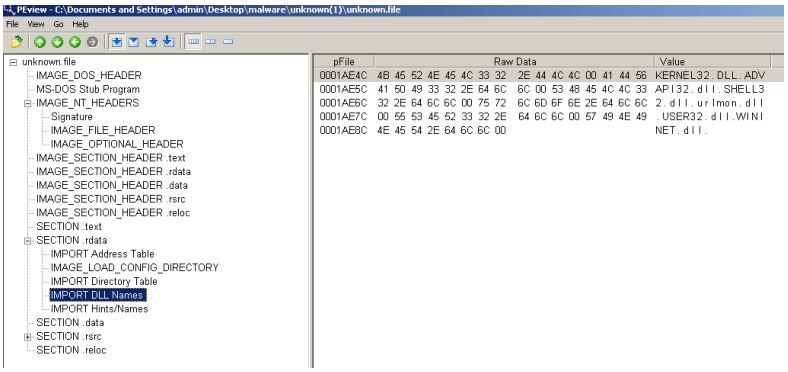


Figure.2.1.3.2.5 - PEview DLL Names.

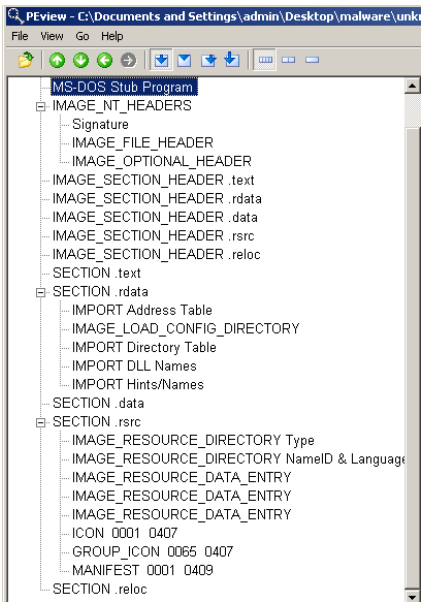


Figure.2.1.3.2.6 - The architecture is now different in PEview

Initially I had confusion about running the file as the system didn't have any programs to run it, so I changed the file extension to .exe, this didn't work, so I changed the file name to pdf as once it had changed to .exe it had the adobe logo, this also didn't work. I checked the file against Virus Total to see if this would tell me, this shows the file is a .exe, additionally, when changed to .exe the logo changes and the type changes to application.

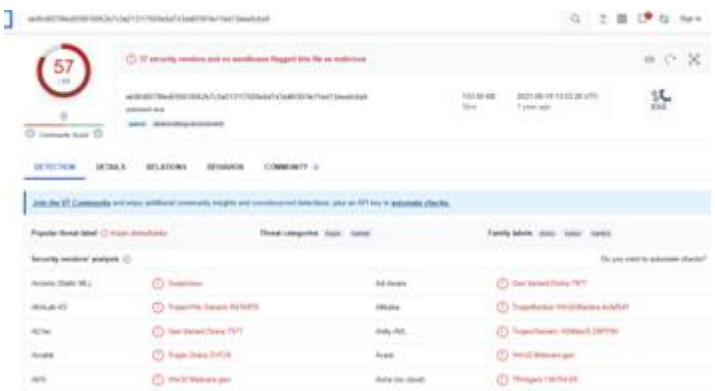


Figure.2.1.3.2.7 – VirusTotal output

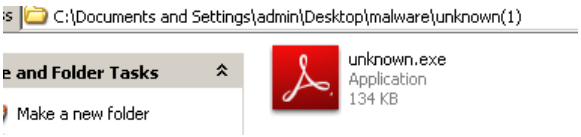


Figure.2.1.3.2.8 - Image of change of file to .exe, note the logo and type change.

It is at this point I realise that not all malware is obvious, so I proceeded to reboot to a clean snapshot, then start RegShot and compare to see if anything was happening in the background. Even though it looked like nothing had happened, in the background, unknown.exe was adding files, deleting files, adding keys, etc. I also noticed that a lot of the added and deleted files where the extension .py, this points to the executable code being python. Please see the html file in the index for the first RegShot of unknown.

Figure.2.1.3.2.9 – RegShot Comparison

2.1.3.3 – Network Analysis of 'unknown' File

After my analysis, and discovery of the file extension, I changed it to 'known.exe' and attempted to open it, nothing happened, luckily I have captured my 1st shot with RegShot and was now ready to capture the second, which showed multiple changes within the system upon executing the file, a noted 27,738 with remote executable peeping out among the thousands of lines of changes, I switched back to a clean snapshot I used Procmon to monitor activity, this corroborated my findings. With network activity, the right tool to use was RegShot and Procmon, Wireshark showed a little activity which seemed suspicious. Procmon showed multiple executable processes happening such as 'CreateFile' and 'RegOpenKey', I filtered the results to show only results from 'unknown.exe', and FakeNet noted a DNS query received and sent.

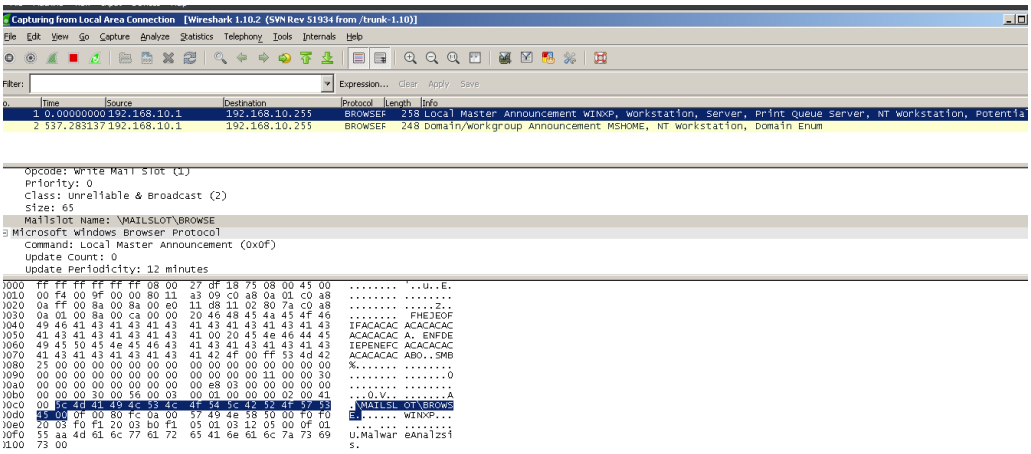


Figure.2.1.3.3.1 – Wireshark Capture for 'unknown' sample

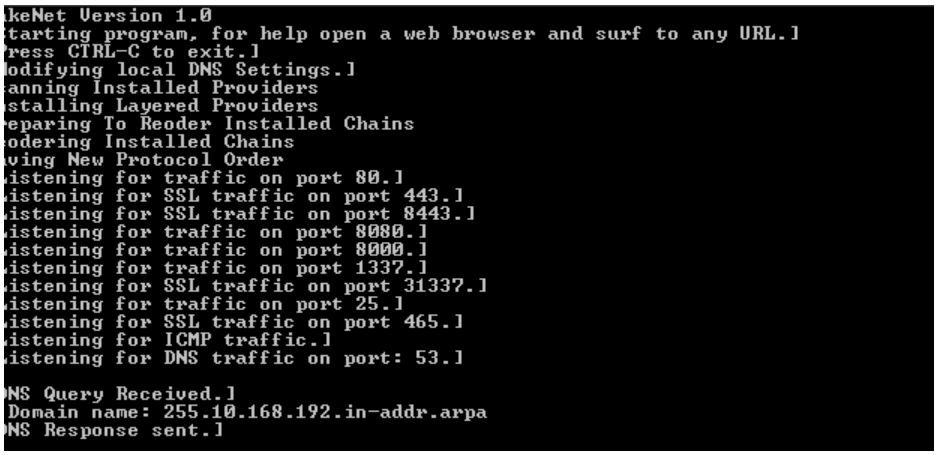


Figure.2.1.3.3.2 – FakeNet Capture for 'unknown' sample

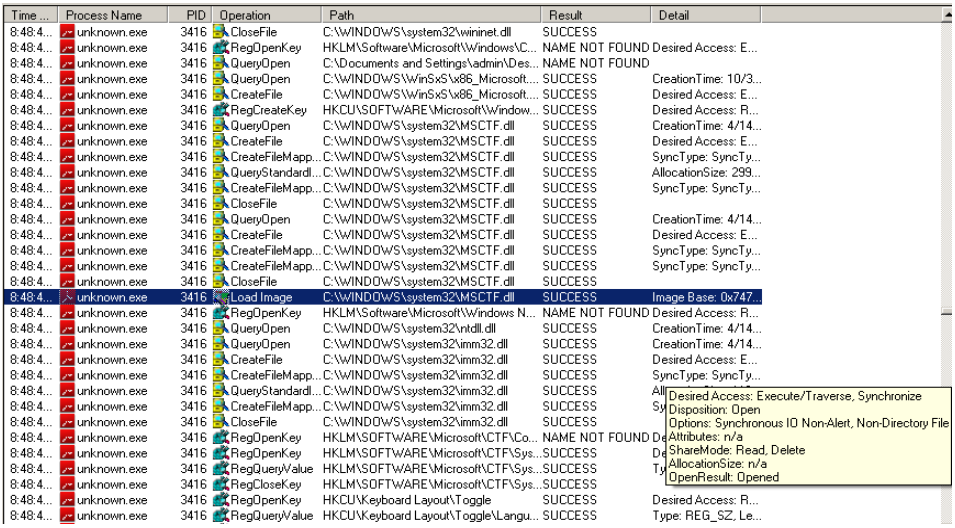


Figure.2.1.3.3.3 – Procmon Capture for 'unknown' sample

3 - Analysis & Reverse Engineering of a Malicious DLL

3.1 - Scenario & Goal

A work colleague came across an email with an attachment and decided to open it, they are now concerned that their workstation may be infected. As requested, I have carried out Basic and Dynamic Static Analysis of the file to see what we are dealing with and to make a plan with how to proceed and protect/isolate the workstation.

3.2 - Environment & Tools

I am using WindowsXP in a Virtual Machine setting in VirtualBox.

PEview	RegShot	PEiD
IDAPro	FakeNet	MiTecEXE
FileAlyzer	Wireshark	Procmon

Table.3.2.1 – Tools used for 'malsample.dll'

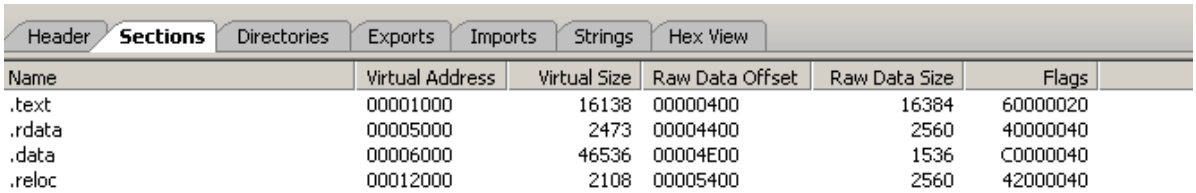
3.3 - Basic Static Analysis of 'malsample.dll'

MiTecEXE

When checking the dll file in MitecEXE I noticed that on the sections tab under '.data. the virtual size was much larger than the raw data size, meaning it could be packed. It also has multiple imports and exports. In this case, the DLL sample has five exports: Install, ServiceMain, UninstallService, installA, and uninstallA. These exports suggest that the sample may be related to a Windows service, as these are common names for service-related functions. The Install and UninstallService functions suggest that the sample may be designed to install or uninstall a service, while the ServiceMain function is the entry point for a service. The installA and uninstallA functions may be alternate versions of the install and uninstall functions that use ASCII strings rather than Unicode strings.

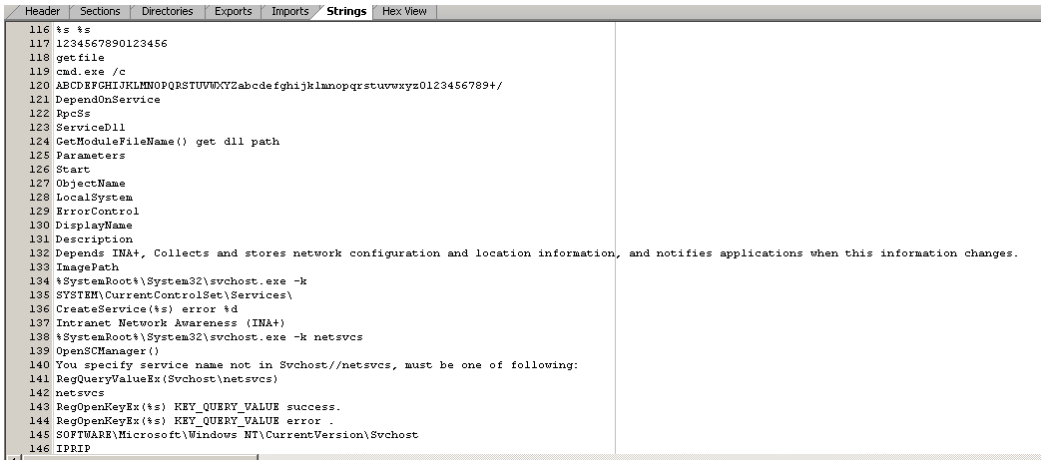
The DLL sample has imports from five different libraries: ADVAPI.dll, KERNEL32.dll, MSVCRT.dll, WININET.dll, and WS2_32.dll. The ADVAPI.dll library is commonly used for handling Windows services and security functions, which supports the hypothesis that the DLL is related to a Windows service. The KERNEL32.dll library provides basic functions for Windows operating systems, while MSVCRT.dll is the Microsoft Visual C++ Runtime Library that provides support for C++ programs. The WININET.dll library is used for internet-related functions, while the WS2_32.dll library provides support for network-related functions.

The exports and imports of this DLL sample suggest that it may be related to a Windows service that involves network and/or internet functionality. However, further analysis is necessary to fully understand the behavior and purpose of the sample. When investigating the Strings Tab, there are many function calls that could be legitimate so it's important for me to conduct Dynamic Analysis to see how this sample operates when it is executed.


Header | **Sections** | Directories | Exports | Imports | Strings | Hex View

Name	Virtual Address	Virtual Size	Raw Data Offset	Raw Data Size	Flags
.text	00001000	16138	00000400	16384	60000020
.rdata	00005000	2473	00004400	2560	40000040
.data	00006000	46536	00004E00	1536	C0000040
.reloc	00012000	2108	00005400	2560	42000040

Figure.3.3.1 – MiTec EXE Sections Tab Information


Header | Sections | Directories | Exports | Imports | **Strings** | Hex View

```
116 %s %s
117 1234567890123456
118 getfile
119 cmd.exe /c
120 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz01234567890+/
121 DependOnService
122 RpcSs
123 ServiceDll
124 GetModuleFileName() get dll path
125 Parameters
126 Start
127 ObjectName
128 LocalSystem
129 ErrorControl
130 DisplayName
131 Description
132 Depends INA+, Collects and stores network configuration and location information, and notifies applications when this information changes.
133 ImagePath
134 %SystemRoot%\System32\svchost.exe -k
135 SYSTEM\CurrentControlSet\Services\
136 CreateService(%s) error %d
137 Intranet Network Awareness (INA+)
138 %SystemRoot%\System32\svchost.exe -k netsvcs
139 OpenSCManager()
140 You specify service name not in Svchost//netsvcs, must be one of following:
141 RegQueryValueEx(Svchost\netsvcs)
142 netsvcs
143 RegOpenKeyEx(%s) KEY_QUERY_VALUE success.
144 RegOpenKeyEx(%s) KEY_QUERY_VALUE error .
145 SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost
146 IPRIP
```

Figure.3.3.2 – MiTec EXE Strings Tab Information

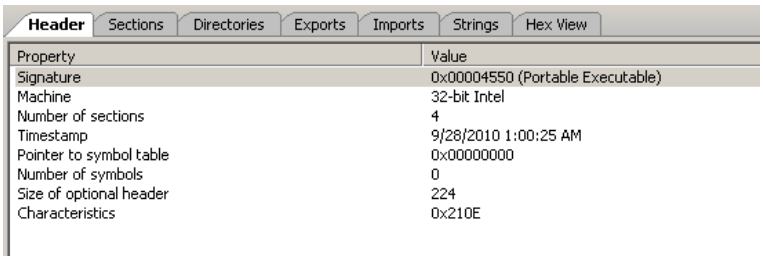


Figure.3.3.3 – MiTec EXE Header Tab Information

PEiD

The presence of an overlay may be an indication that the file has been tampered with as legitimate files typically do not have overlays. However, it is also possible that the overlay was added by a legitimate program or by a benign tool used by the malware author, so the presence of an overlay alone is not sufficient to determine whether a file is malicious. Additional analysis is necessary to determine whether a file is malicious.

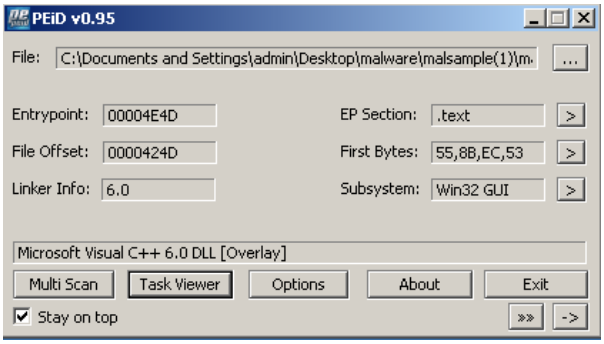


Figure.3.3.3 – PEiD Information

Figure.3.3.4 - PEiD System Tasks Information

FileAlyzer

This section tells me that this was created in 2011 and confirms the overlay.

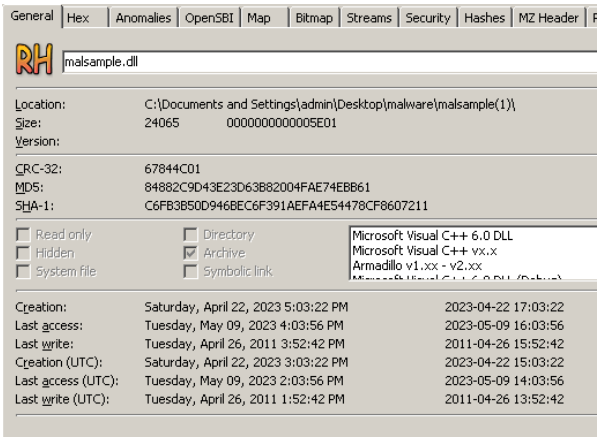


Figure.3.3.5 - FileAlyzer General Tab Information

In the hex tab, you can see the message; this program cannot be done in dos mode.

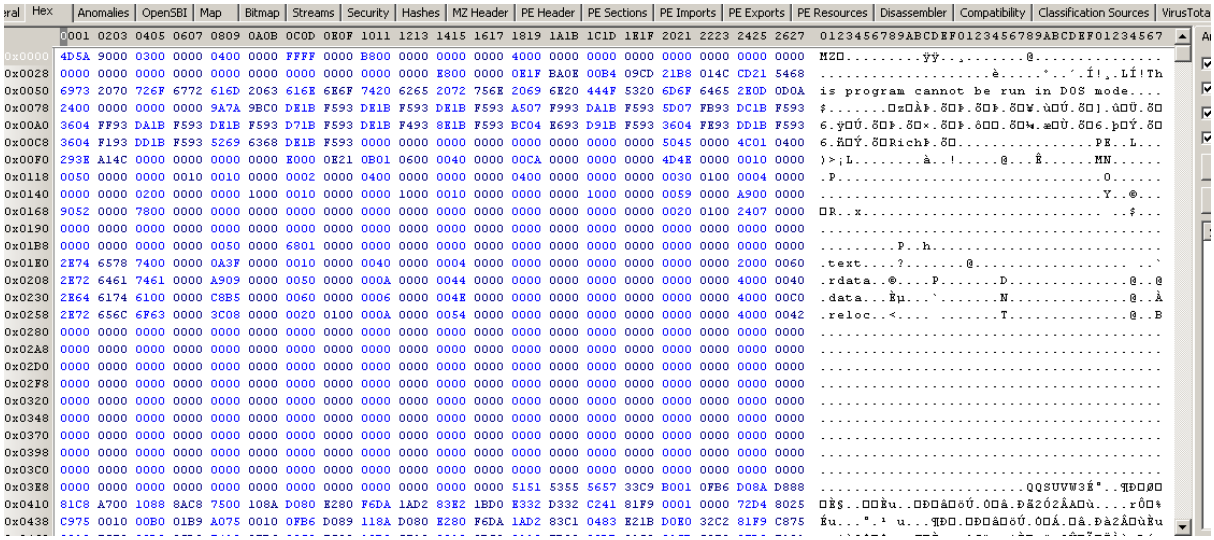


Figure.3.3.6 - FileAlyzer Hex Tab

The OpenSBI tab notes the file size could be incorrect.

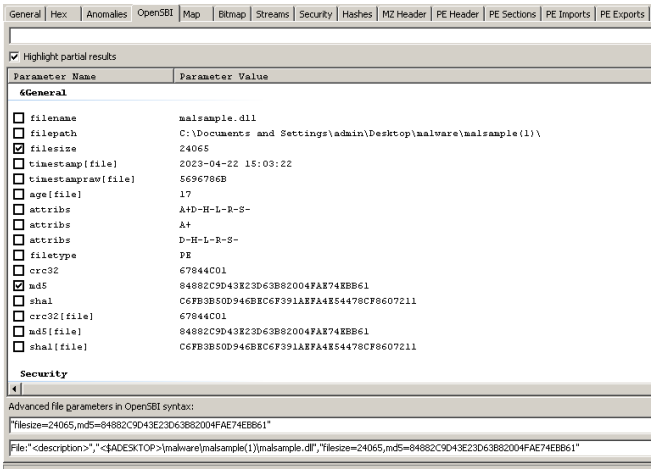


Figure.3.3.7 - FileAlyzer OpenSBI

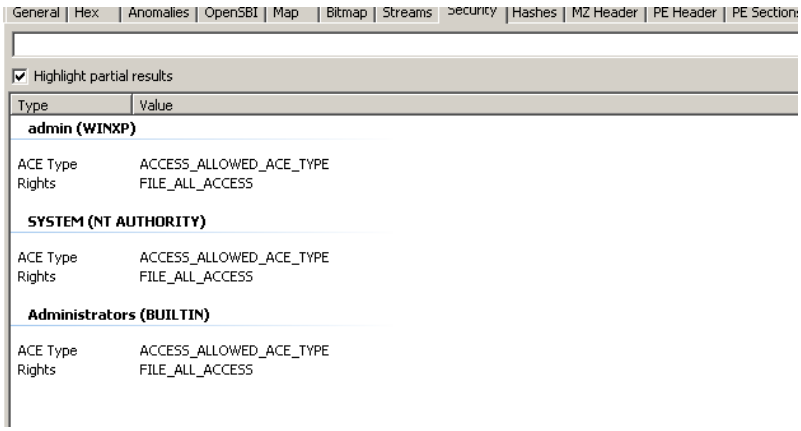


Figure.3.3.8 - FileAlyzer Security

Here the Security tabs shows an executable in the characteristics.

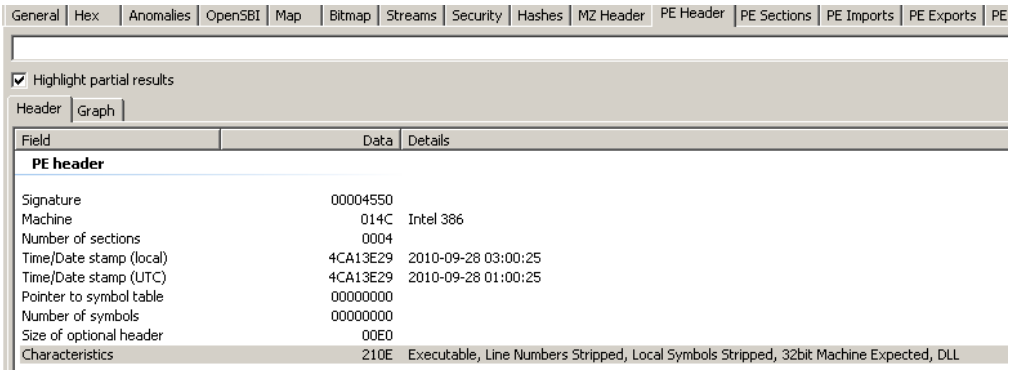


Figure.3.3.9 - FileAlyzer PE Header Tab

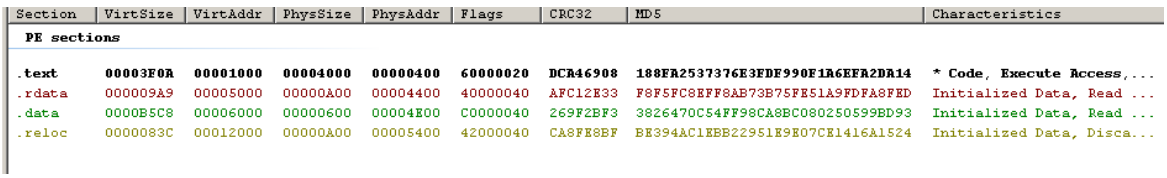


Figure.3.3.10(a) - FileAlyzer PE Sections Tab

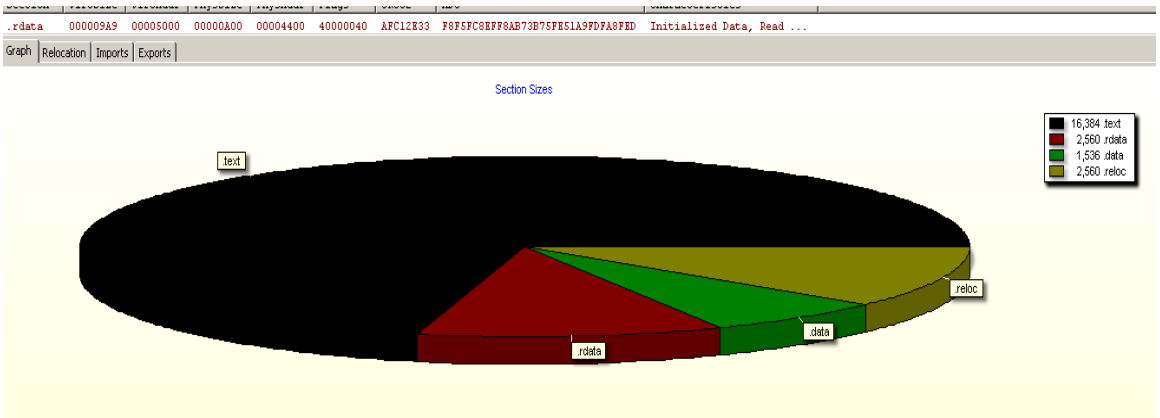


Figure.3.3.10(b) - FileAlyzer PE Sections Graph Tab

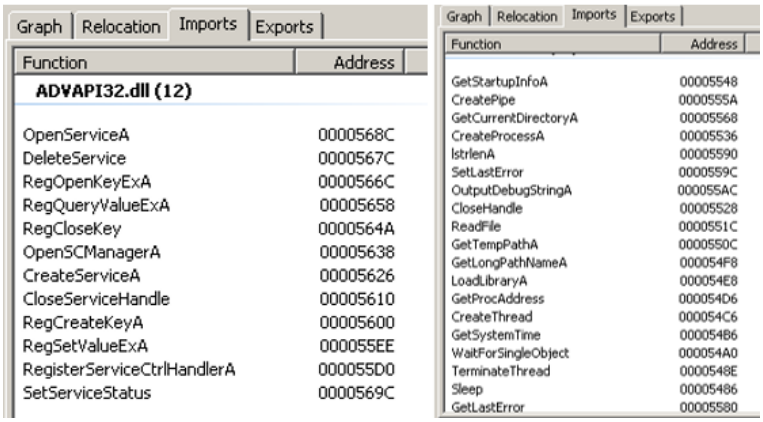


Figure.3.3.10(c) - FileAlyzer PE Sections Imports Tab (ADVAPI(L) & KERNEL32(R))

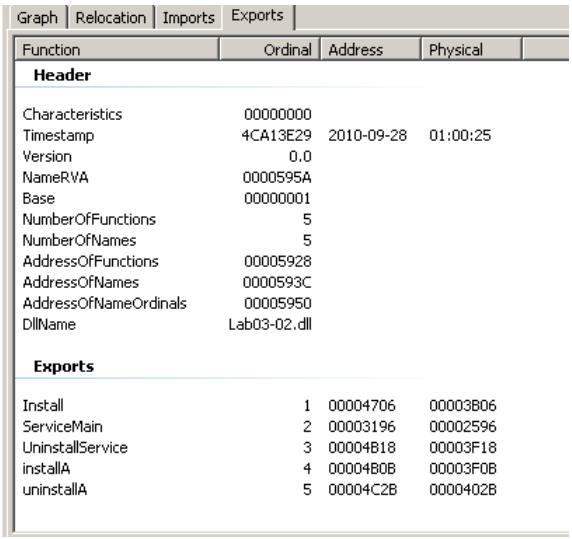


Figure.3.3.10(d) - FileAlyzer PE Sections Exports Tab

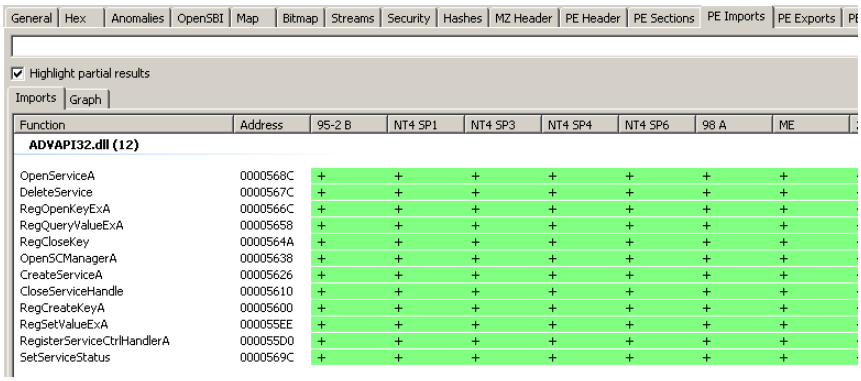


Figure.3.3.11(a) - FileAlyzer PE Imports Tab ADVAPI32.dll

KERNEL32.dll (20)

GetStartupInfoA	00005548	+	+	+	+	+	+	+	+
CreatePipe	0000555A	+	+	+	+	+	+	+	+
GetCurrentDirectoryA	00005568	+	+	+	+	+	+	+	+
CreateProcessA	00005536	+	+	+	+	+	+	+	+
lstrlenA	00005590	+	+	+	+	+	+	+	+
SetLastError	0000559C	+	+	+	+	+	+	+	+
OutputDebugStringA	000055AC	+	+	+	+	+	+	+	+
CloseHandle	00005528	+	+	+	+	+	+	+	+
ReadFile	0000551C	+	+	+	+	+	+	+	+
GetTempPathA	0000550C	+	+	+	+	+	+	+	+
GetLongPathNameA	000054F8	-	-	-	-	-	+	+	+
LoadLibraryA	000054E8	+	+	+	+	+	+	+	+
GetProcAddress	000054D6	+	+	+	+	+	+	+	+
CreateThread	000054C6	+	+	+	+	+	+	+	+
GetSystemTime	000054B6	+	+	+	+	+	+	+	+
WaitForSingleObject	000054A0	+	+	+	+	+	+	+	+
TerminateThread	0000548E	+	+	+	+	+	+	+	+
Sleep	00005486	+	+	+	+	+	+	+	+
GetLastError	00005580	+	+	+	+	+	+	+	+
GetModuleFileNameA	00005470	+	+	+	+	+	+	+	+

Figure.3.3.11(b) - FileAlyzer PE Imports Tab KERNELL32.dll

General | Hex | Anomalies | OpenSBI | Map | Bitmap | Streams | Security | Hashes | M

Highlight partial results

Function	Ordinal	Address	Physical
Header			
Characteristics	00000000		
Timestamp	4CA13E29	2010-09-28	01:00:25
Version	0.0		
NameRVA	0000595A		
Base	00000001		
NumberOfFunctions	5		
NumberOfNames	5		
AddressOfFunctions	00005928		
AddressOfNames	0000593C		
AddressOfNameOrdinals	00005950		
DllName	Lab03-02.dll		
Exports			
Install	1	00004706	00003B06
ServiceMain	2	00003196	00002596
UninstallService	3	00004B18	00003F18
installA	4	00004B0B	00003F0B
uninstallA	5	00004C2B	0000402B

Figure.3.3.12 - FileAlyzer PE Exports Tab

Some interesting points here, including an import of functions.

General | Hex | Anomalies | OpenSBI | Map | Bitmap | Streams | Security | Hashes | MZ Header | PE Header | PE Sections | PE Imports | PE Exports | PE Resources | Disassemble

Address: 0x00004285 Disassemble

Relocated	Physical	Bytecode	Assembler	Comments
0x100036DA	0x000042DA	53	PUSH EBX	
0x100036DB	0x000042DB	FFD0	CALL EAX	
0x100036DD	0x000042DD	89450C	MOV [EBP+0C], EAX	
0x100036E0	0x000042E0	8B450C	MOV EAX, [EBP+0C]	
0x100036E3	0x000042E3	5F	POP EDI	
0x100036E4	0x000042E4	5E	POP ESI	
0x100036E5	0x000042E5	5B	POP EBX	
0x100036E6	0x000042E6	5D	POP EBP	
0x100036E7	0x000042E7	C20C00	RET 000C	
0x100036EA	0x000042EA	FF25A0500010	JMP [100050A0]	TODO: function from import list (0xFF /4)!
0x100036F0	0x000042F0	FF2598500010	JMP [10005098]	TODO: function from import list (0xFF /4)!
0x100036F6	0x000042F6	CC	INT 3	
0x100036F7	0x000042F7	CC	INT 3	
0x100036F8	0x000042F8	CC	INT 3	
0x100036F9	0x000042F9	CC	INT 3	
0x100036FA	0x000042FA	CC	INT 3	
0x100036FB	0x000042FB	CC	INT 3	
0x100036FC	0x000042FC	CC	INT 3	
0x100036FD	0x000042FD	CC	INT 3	
0x100036FE	0x000042FE	CC	INT 3	
0x100036FF	0x000042FF	CC	INT 3	
0x10003700	0x00004300	B838520010	MOV EAX, 10005238	
0x10003705	0x00004305	E972FEFFFF	JMP +0000417C	Calling near: 1000417C
0x1000370A	0x0000430A	0000	ADD [EAX], AL	

Figure.3.3.13 - FileAlyzer Disassembler Tab

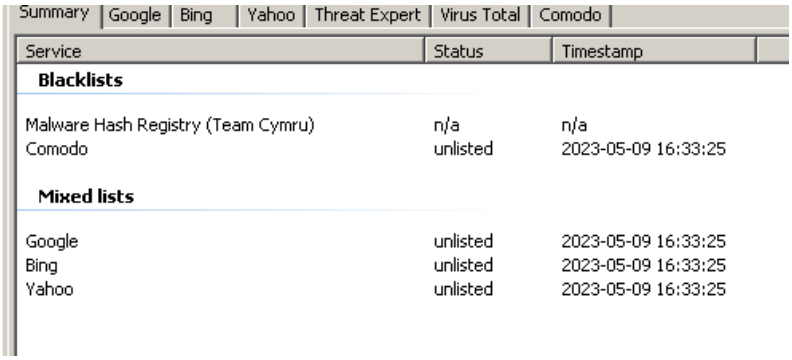


Figure.3.3.14 - FileAlyzer Classification Tab

3.4 – Dynamic Analysis of 'malsample.dll'

To execute the file I had to engage in command line actions by unpacking it, I used the command 'upx -d malsample.dll', it's important to note here that the sample had to be in the same directory as the upx tool. Using Regshot I analysed and compared before and after execution to see if this affects the system in anyway. There are 10 modified values, 6 added files, 1 deleted file, 3 modified files/attributes, 1 folder added then deleted, in total, there's 48 changes.

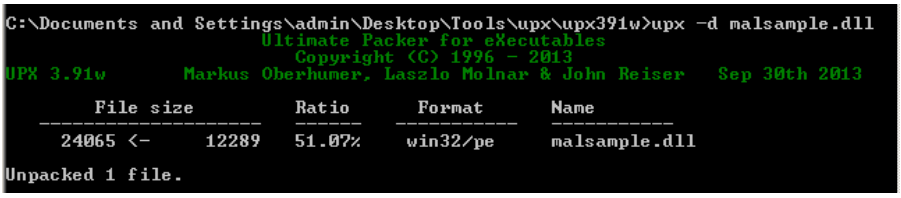


Figure.3.4.1 – Unpacked malsample

RegShot

Here I have ran the service 'install' and documented the findings. Command: *run32dll.exe malsample.dll,install*

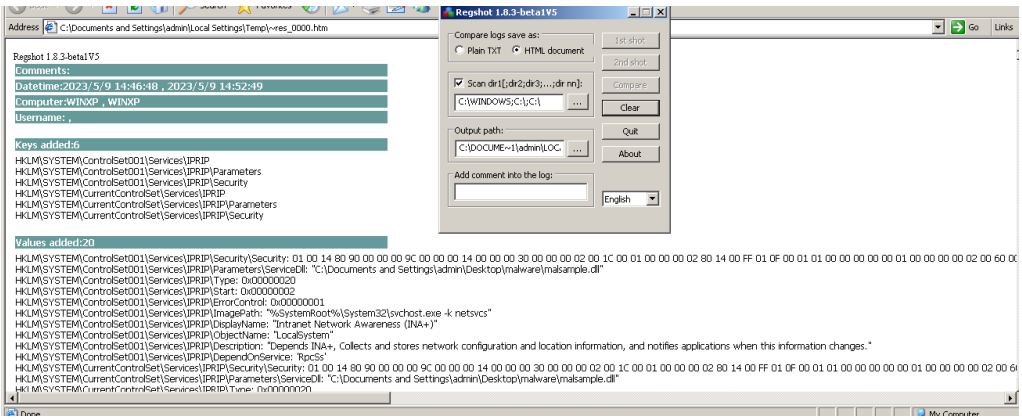


Figure.3.4.2 - RegShot Capture of 'malsample.dll' 1

```
 Directory of C:\Documents and Settings\admin\Desktop\malware

04/22/2023  05:45 PM    <DIR>          .
04/22/2023  05:45 PM    <DIR>          ..
04/20/2023  10:04 PM    <DIR>          cw_pdf_files(1)
04/19/2023  07:26 PM           224,078 cw_pdf_files(1).7z
05/09/2023  04:19 PM    <DIR>          malsample(1)
04/19/2023  07:27 PM            10,418 malsample(1).7z
04/22/2023  08:32 PM    <DIR>          unknown(1)
04/19/2023  07:26 PM            51,938 unknown(1).7z
04/22/2023  05:45 PM    <DIR>          unknownfilesnotes
04/20/2023  10:07 PM    <DIR>          wireshark
               3 File(s)        286,434 bytes
               7 Dir(s)   5,050,109,952 bytes free

C:\Documents and Settings\admin\Desktop\malware>rundll32.exe malsample.dll, Inst
allA

C:\Documents and Settings\admin\Desktop\malware>rundll32.exe malsample.dll, Inst
all

C:\Documents and Settings\admin\Desktop\malware>rundll32.exe malsample.dll, inst
all

C:\Documents and Settings\admin\Desktop\malware>
```

Figure.3.4.3 – Command Line of 'malsample.dll'

FakeNet

```
[Listening for traffic on port 80.]
[Listening for SSL traffic on port 443.]
[Listening for SSL traffic on port 8443.]
[Listening for traffic on port 8080.]
[Listening for traffic on port 8000.]
[Listening for traffic on port 1337.]
[Listening for SSL traffic on port 31337.]
[Listening for SSL traffic on port 465.]
[Listening for traffic on port 25.]
[Listening for ICMP traffic.]
[Listening for DNS traffic on port: 53.]

[DNS Query Received.]
  Domain name: www.wireshark.org
[DNS Response sent.]


[Received new connection on port: 443.]
[New request on port 443 with SSL.]
  [Received unsupported HTTP request.]

[DNS Query Received.]
  Domain name: 255.10.168.192.in-addr.arpa
[DNS Response sent.]
```

Figure.3.4.5– FakeNet Monitor Output

```
Command Prompt
05/09/2023  06:51 PM    <DIR>          .
05/09/2023  06:51 PM    <DIR>          ..
05/09/2023  06:51 PM    <DIR>          malsample
04/19/2023  07:27 PM            10,418 malsample(1).7z
               1 File(s)         10,418 bytes
               3 Dir(s)   5,050,105,856 bytes free

C:\Documents and Settings\admin\Desktop\malsample>cd malsample

C:\Documents and Settings\admin\Desktop\malsample\malsample>dir
 Volume in drive C has no label.
 Volume Serial Number is C009-159C

 Directory of C:\Documents and Settings\admin\Desktop\malsample\malsample

05/09/2023  06:51 PM    <DIR>          .
05/09/2023  06:51 PM    <DIR>          ..
04/26/2011  03:52 PM            24,065 malsample.dll
               1 File(s)         24,065 bytes
               2 Dir(s)   5,050,105,856 bytes free

C:\Documents and Settings\admin\Desktop\malsample\malsample>rundll32.exe malsamp
le.dll install

C:\Documents and Settings\admin\Desktop\malsample\malsample>
```

Figure.3.4.6 – dll run command

Process Explorer

No additional services seemed to have started, peep the before and after below. I did note that when investigated further within the Internet Explorer running process, the same strings were present in the strings tab of this process within Process Explorer.

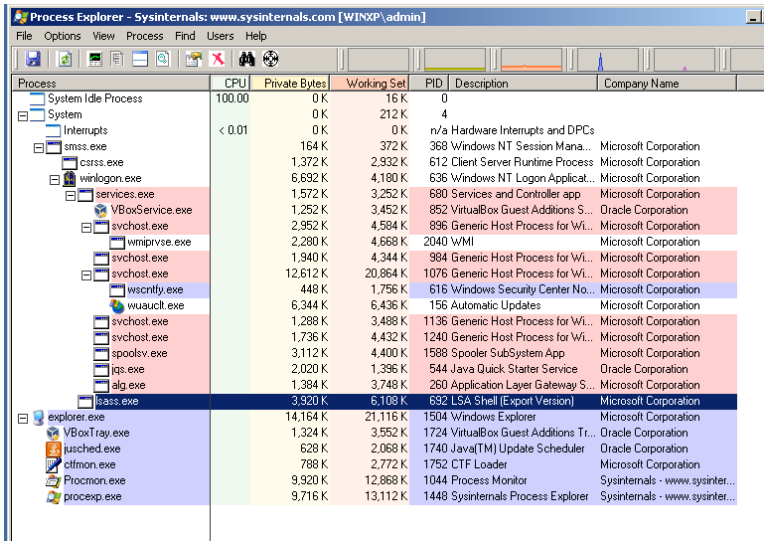


Figure.3.4.7 – Process Explore Prior to Execution

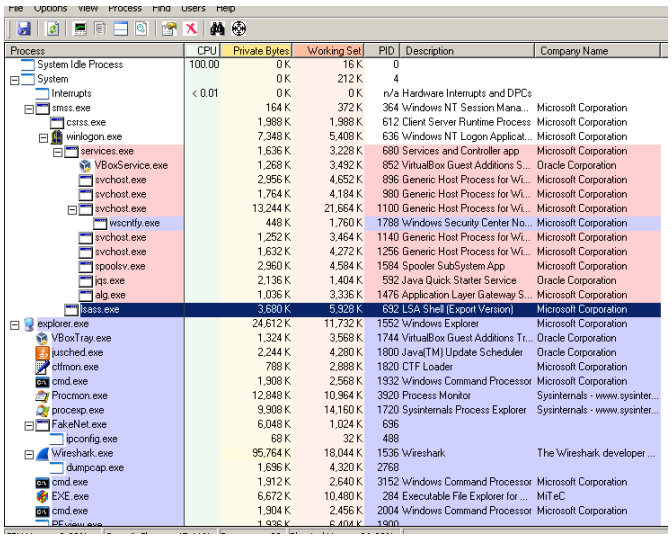


Figure.3.4.8 – Process Explore After to Execution

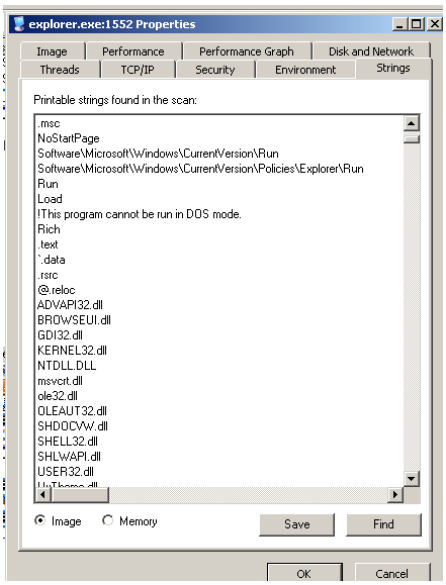
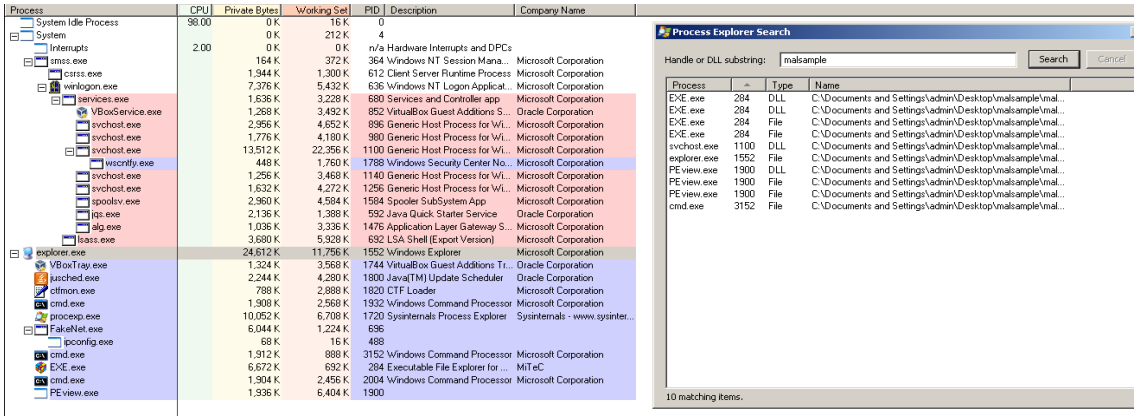


Figure.3.4.9 – Process Explore Internet Explorer Properties, then Strings tab

At this point I carried out a search for the malsample DLL's within Process Explorer which would give us the address of the process that's operating under this sample, it is seemingly operating under a known SVHost process to obfuscate itself:



Process Monitor

Process Monitor shows all of the services operating under the rundll32.exe process which match up to the service names in the analysis stage that were discovered.

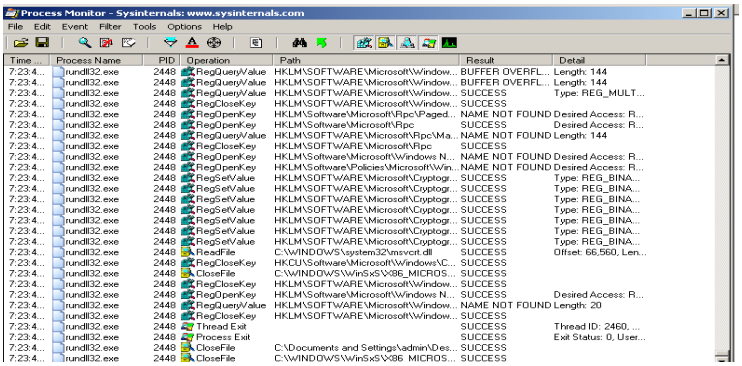


Figure.3.4.10 – Process Monitor Filtered to rundll32.exe

3.5 – Network Analysis of 'malsample.dll'

The network activity after using the rundll32.exe malsample.dll,install corroborates the information found with the analysis stage, it seems to have network capabilities from looking at the strings, and wireshark confirms this. There's also a log that has the same words 'Malware Analzsis' discovered in the analysis stage.

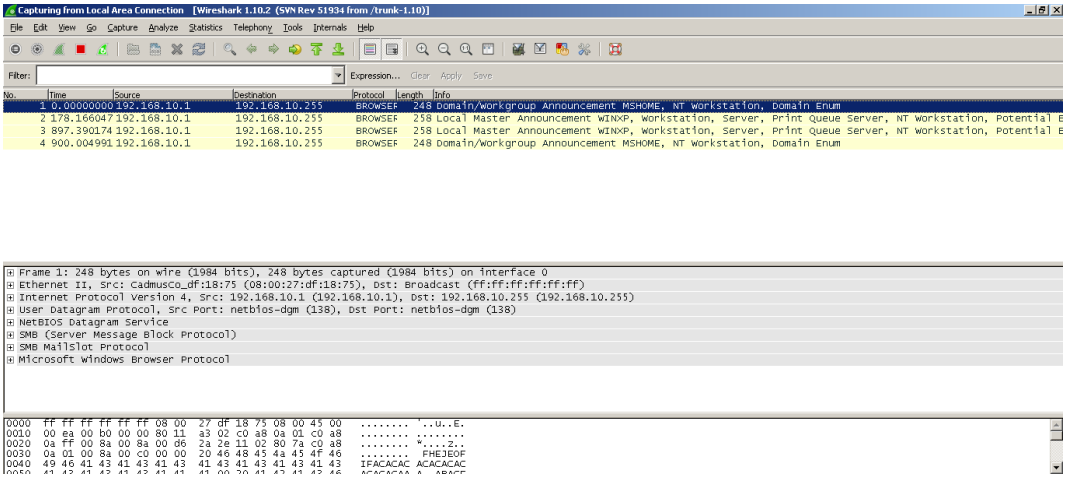


Figure.3.5.1 – Wireshark after installing service (1)

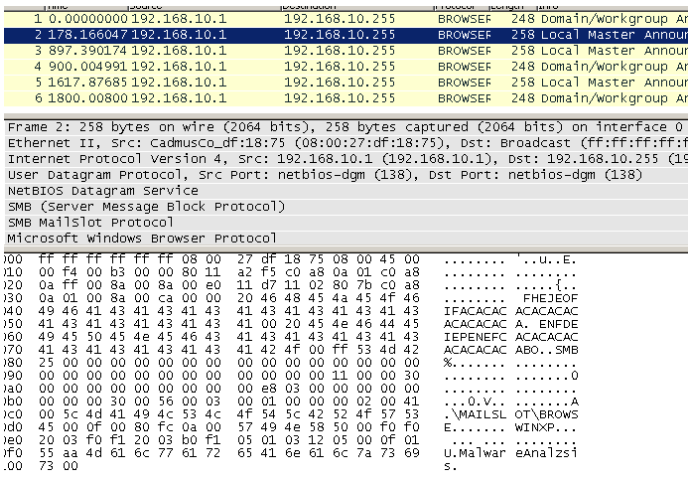


Figure.3.5.2 – Wireshark after installing service (2)

3.6 – Reverse Engineering of 'malsample.dll'

(a) How many functions are exported by the DLL? There are six exported functions; Install, ServiceMain, UninstallService, InstallA, and DllEntryPoint.

(b) What are the addresses of the functions that the DLL exports?

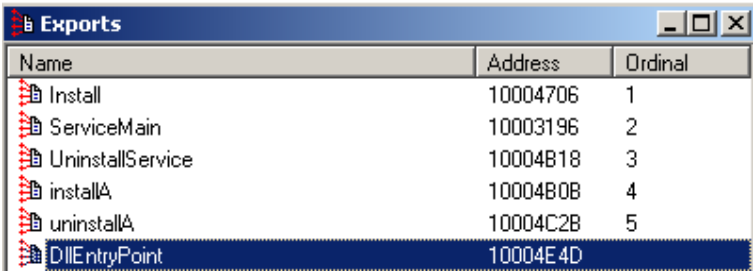


Figure.3.6.1 – malsample.dll Exports and Addresses (IDA)

(c) How many functions call the kernel32 API LoadLibrary? I documented twenty occurrences that kernel32.dll LoadLibrary is called.

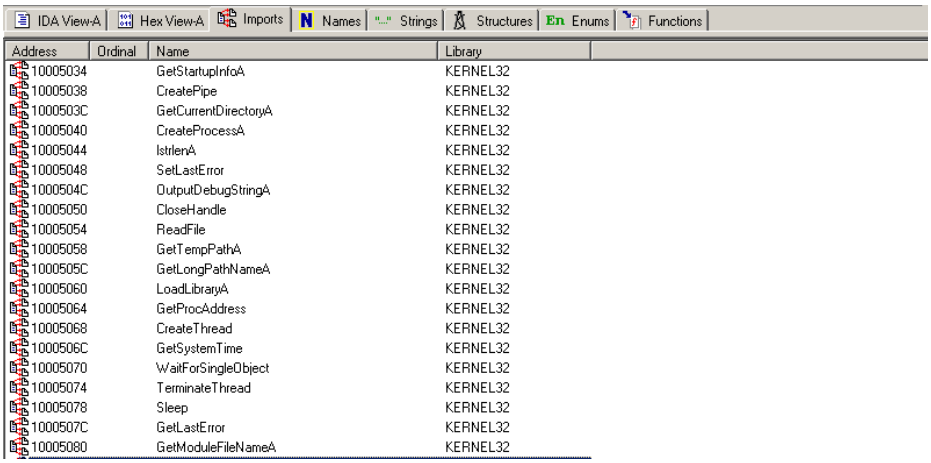


Figure.3.6.2 – kernel32.dll LoadLibrary (IDA)

(d) How many times is the kernel32 API Sleep() called in the DLL? (support your answers with documentary evidence, e.g., screenshots). The API Sleep() is called once.

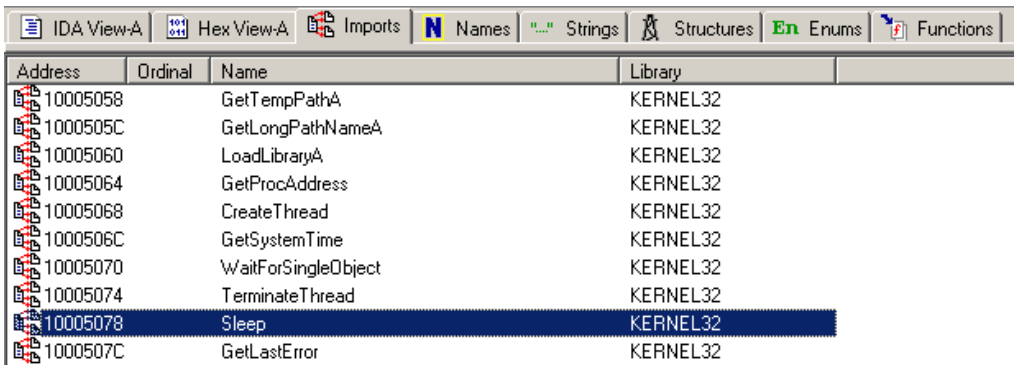


Figure.3.6.3 – kernel32 API Sleep() (IDA)

5. Navigate to the ServiceMain function.

(a) Show the graph view of the function

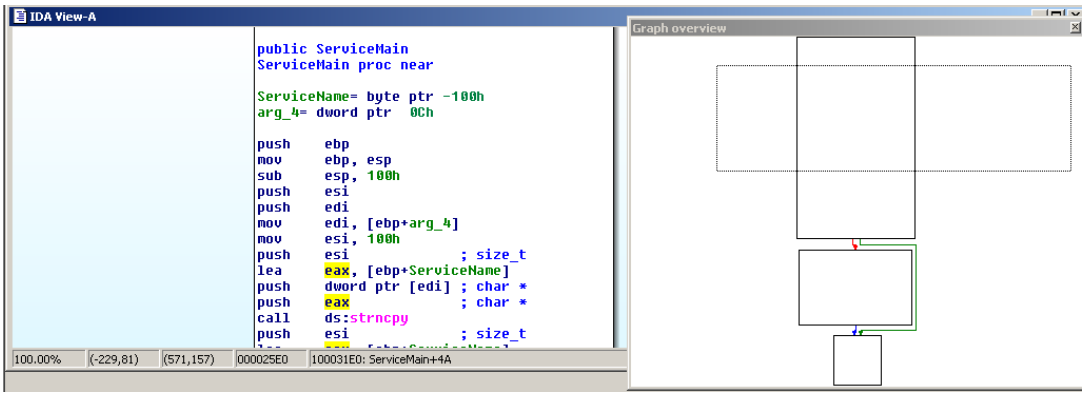


Figure.3.6.4 – Graph View of ServiceMain (IDA)

(b) The main subroutine (of the ServiceMain function) jumps to a location where the code calls the kernel32 API Sleep() right after the JZ assembly instruction. What is the value of the parameter used by this Sleep() call?

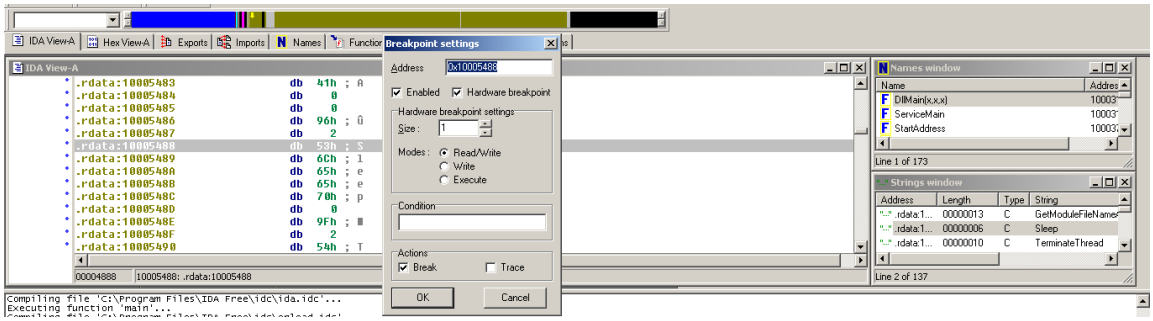


Figure.3.6.4(a) – Answer to question 5b

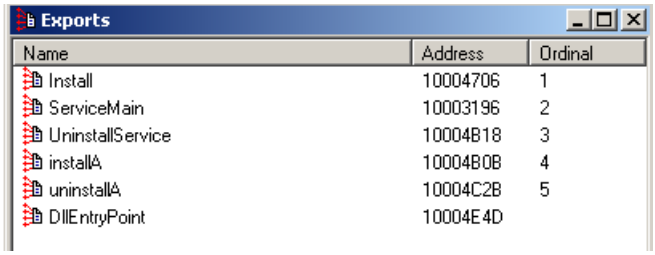


Figure.3.6.5 – Exports (IDA)

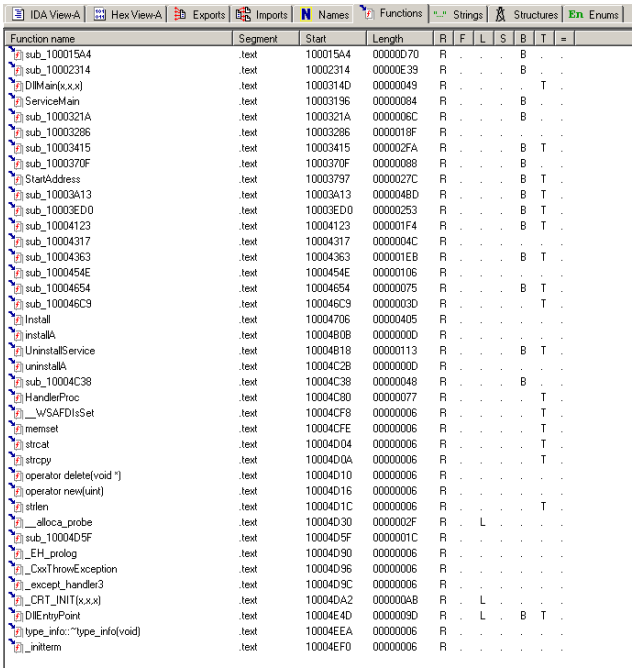
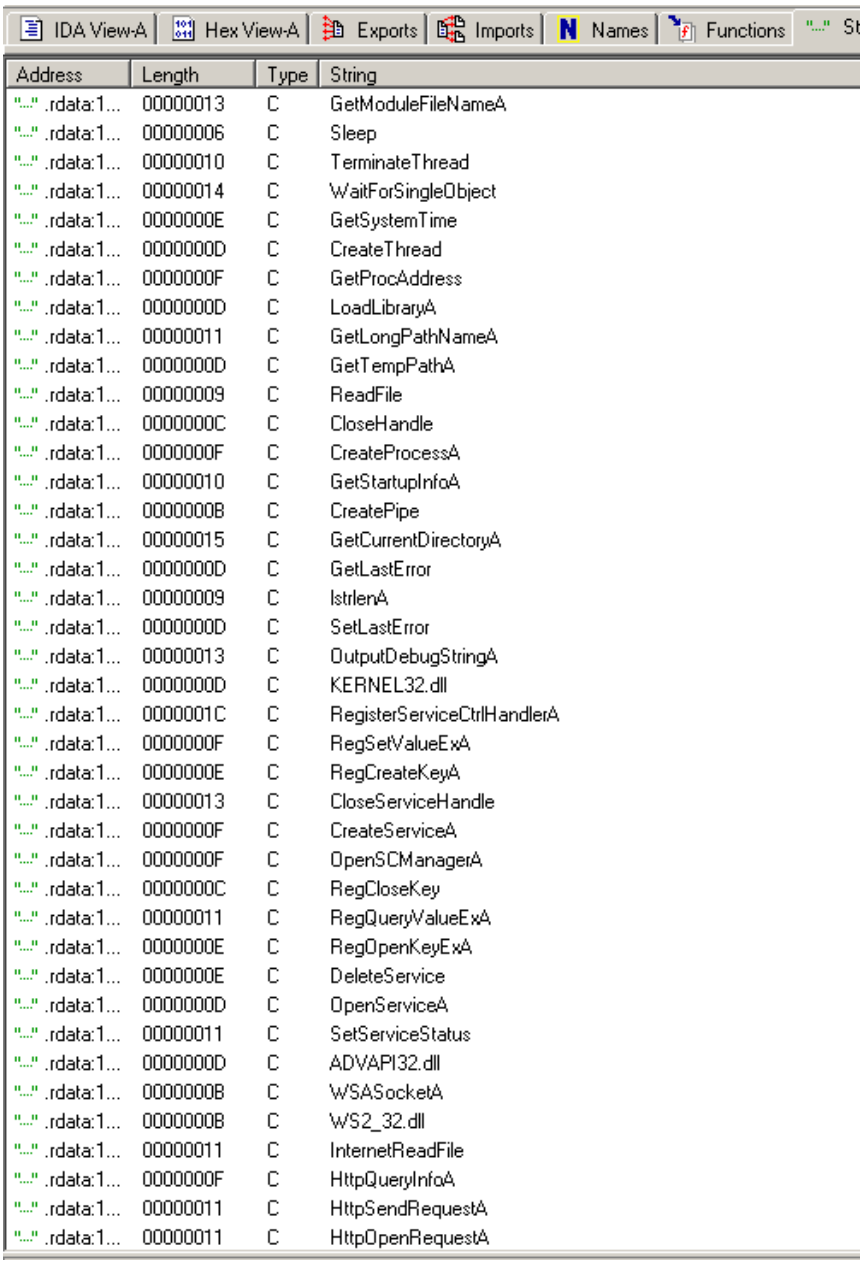


Figure.3.6.6 – Functions (IDA)



IDA View-A | Hex View-A | Exports | Imports | Names | Functions | "..." St

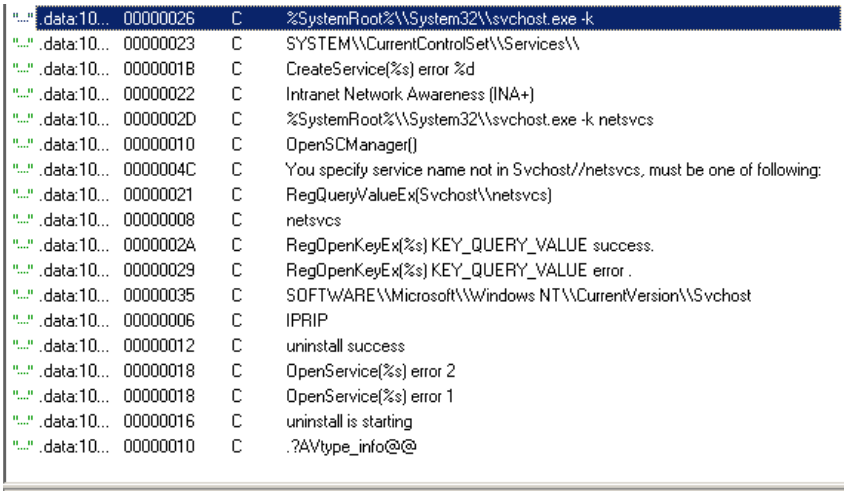
Address	Length	Type	String
"..." .rdata:1...	00000013	C	GetModuleFileNameA
"..." .rdata:1...	00000006	C	Sleep
"..." .rdata:1...	00000010	C	TerminateThread
"..." .rdata:1...	00000014	C	WaitForSingleObject
"..." .rdata:1...	0000000E	C	GetSystemTime
"..." .rdata:1...	0000000D	C	CreateThread
"..." .rdata:1...	0000000F	C	GetProcAddress
"..." .rdata:1...	0000000D	C	LoadLibraryA
"..." .rdata:1...	00000011	C	GetLongPathNameA
"..." .rdata:1...	0000000D	C	GetTempPathA
"..." .rdata:1...	00000009	C	ReadFile
"..." .rdata:1...	0000000C	C	CloseHandle
"..." .rdata:1...	0000000F	C	CreateProcessA
"..." .rdata:1...	00000010	C	GetStartupInfoA
"..." .rdata:1...	0000000B	C	CreatePipe
"..." .rdata:1...	00000015	C	GetCurrentDirectoryA
"..." .rdata:1...	0000000D	C	GetLastError
"..." .rdata:1...	00000009	C	lstrlenA
"..." .rdata:1...	0000000D	C	SetLastError
"..." .rdata:1...	00000013	C	OutputDebugStringA
"..." .rdata:1...	0000000D	C	KERNEL32.dll
"..." .rdata:1...	0000001C	C	RegisterServiceCtrlHandlerA
"..." .rdata:1...	0000000F	C	RegSetValueExA
"..." .rdata:1...	0000000E	C	RegCreateKeyA
"..." .rdata:1...	00000013	C	CloseServiceHandle
"..." .rdata:1...	0000000F	C	CreateServiceA
"..." .rdata:1...	0000000F	C	OpenSCManagerA
"..." .rdata:1...	0000000C	C	RegCloseKey
"..." .rdata:1...	00000011	C	RegQueryValueExA
"..." .rdata:1...	0000000E	C	RegOpenKeyExA
"..." .rdata:1...	0000000E	C	DeleteService
"..." .rdata:1...	0000000D	C	OpenServiceA
"..." .rdata:1...	00000011	C	SetServiceStatus
"..." .rdata:1...	0000000D	C	ADVAPI32.dll
"..." .rdata:1...	0000000B	C	WSASocketA
"..." .rdata:1...	0000000B	C	WS2_32.dll
"..." .rdata:1...	00000011	C	InternetReadFile
"..." .rdata:1...	0000000F	C	HttpQueryInfoA
"..." .rdata:1...	00000011	C	HttpSendRequestA
"..." .rdata:1...	00000011	C	HttpOpenRequestA

Figure.3.6.7 – Strings (IDA)

IDA View-A | Hex View-A | Exports | Imports | Names | Functions | Strings | Structures

Address	Length	Type	String
".." .rdata:1...	00000011	C	UninstallService
".." .rdata:1...	00000009	C	installA
".." .rdata:1...	0000000B	C	uninstallA
".." .data:10...	0000000D	C	Y29ubmVjdA==
".." .data:10...	0000001D	C	practicalmalwareanalysis.com
".." .data:10...	0000000B	C	serve.html
".." .data:10...	0000000D	C	dW5zdXBwb3J0
".." .data:10...	00000009	C	c2xlZXA=
".." .data:10...	00000005	C	Y21k
".." .data:10...	00000009	C	cXVpdA==
".." .data:10...	00000011	C	\Windows XP 6.11
".." .data:10...	0000000F	C	CreateProcessA
".." .data:10...	0000000D	C	kernel32.dll
".." .data:10...	00000005	C	.exe
".." .data:10...	00000009	C	HTTP/1.1
".." .data:10...	00000006	C	%s %s
".." .data:10...	00000011	C	1234567890123456
".." .data:10...	00000005	C	quit
".." .data:10...	00000005	C	exit
".." .data:10...	00000008	C	getfile
".." .data:10...	0000000C	C	cmd.exe /c
".." .data:10...	00000041	C	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345...
".." .data:10...	00000005	C	--I>
".." .data:10...	00000005	C	<!--
".." .data:10...	00000005	C	.PAX
".." .data:10...	00000010	C	DependOnService
".." .data:10...	00000006	C	RpcSs
".." .data:10...	0000000B	C	ServiceDll
".." .data:10...	00000021	C	GetModuleFileName() get dll path
".." .data:10...	0000000B	C	Parameters
".." .data:10...	00000005	C	Type
".." .data:10...	00000006	C	Start
".." .data:10...	0000000B	C	ObjectName
".." .data:10...	0000000C	C	LocalSystem
".." .data:10...	0000000D	C	ErrorControl
".." .data:10...	0000000C	C	DisplayName
".." .data:10...	0000000C	C	Description
".." .data:10...	0000008B	C	Depends INA+, Collects and stores network configuration and location infor...
".." .data:10...	0000000A	C	ImagePath
".." .data:10...	00000026	C	%SystemRoot%\\System32\\svchost.exe -k

Figure.3.6.8 – Strings 2 (IDA)



".." .data:10...	00000026	C	%SystemRoot%\\System32\\svchost.exe -k
".." .data:10...	00000023	C	SYSTEM\\CurrentControlSet\\Services\\
".." .data:10...	0000001B	C	CreateService(%s) error %d
".." .data:10...	00000022	C	Intranet Network Awareness (INA+)
".." .data:10...	0000002D	C	%SystemRoot%\\System32\\svchost.exe -k netsvcs
".." .data:10...	00000010	C	OpenSCManager()
".." .data:10...	0000004C	C	You specify service name not in Svchost//netsvcs, must be one of following:
".." .data:10...	00000021	C	RegQueryValueEx(Svchost\\netsvcs)
".." .data:10...	00000008	C	netsvcs
".." .data:10...	0000002A	C	RegOpenKeyEx(%s) KEY_QUERY_VALUE success.
".." .data:10...	00000029	C	RegOpenKeyEx(%s) KEY_QUERY_VALUE error .
".." .data:10...	00000035	C	SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Svchost
".." .data:10...	00000006	C	IPRIP
".." .data:10...	00000012	C	uninstall success
".." .data:10...	00000018	C	OpenService(%s) error 2
".." .data:10...	00000018	C	OpenService(%s) error 1
".." .data:10...	00000016	C	uninstall is starting
".." .data:10...	00000010	C	.?AVtype_info@@

Figure.3.6.9 – Strings 3 (IDA)

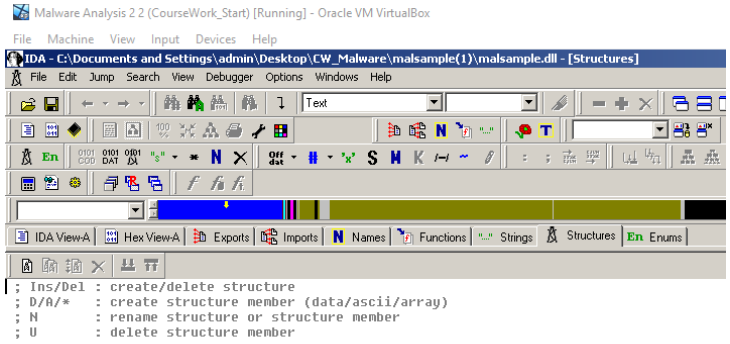


Figure.3.6.10 - Structures (IDA)

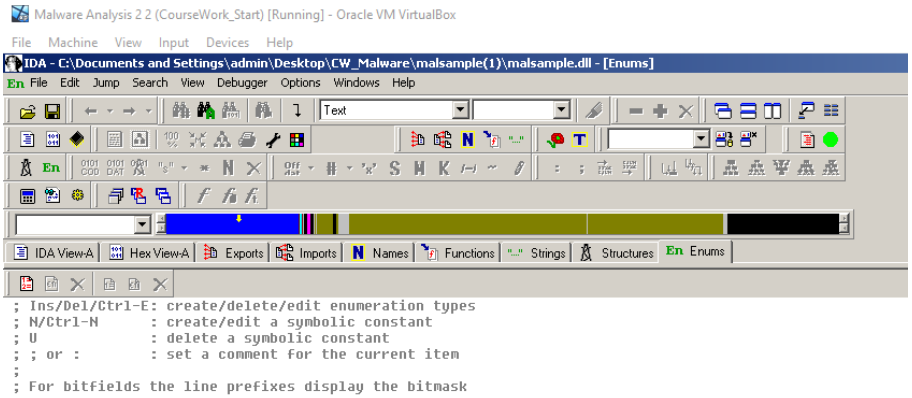


Figure. *3.6.11*– Enums (IDA)

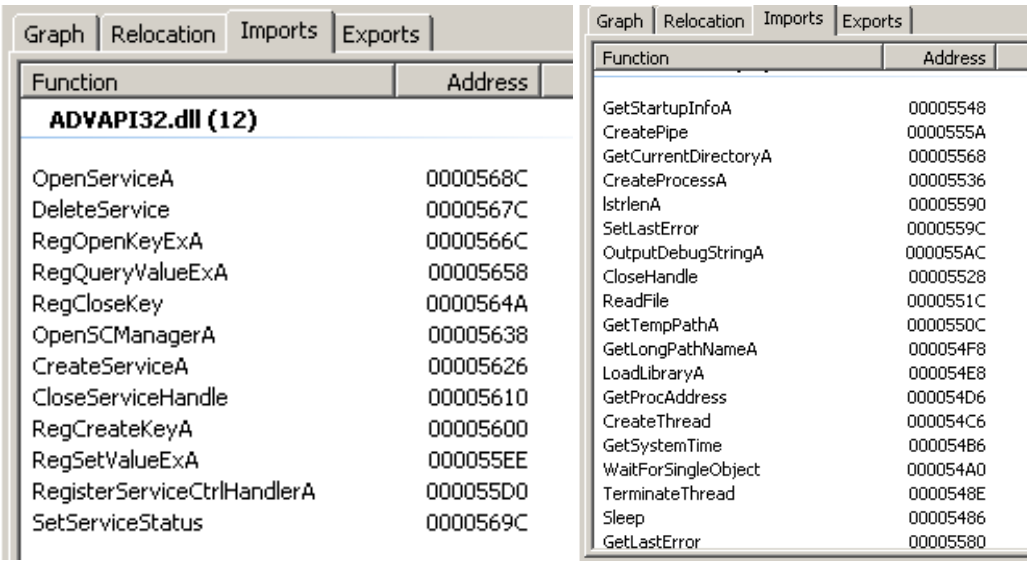


Figure.*3.6.12* – Imports (IDA)

4 – References

11 Best Malware Analysis Tools and Their Features [WWW Document], n.d. URL http://www.varonis.com/blog/malware-analysis-tools (accessed 5.5.23).

Analysis of Malicious Documents- Part 5 [WWW Document], n.d. . Infosec Resour. URL https://resources.infosecinstitute.com/topic/analysis-malicious-documents-part-5/ (accessed 5.5.23).

Arshad, M.M., 2021. Analyzing PDF Files — A Deceitful Malware Specie. Medium. URL https://medium.com/@m01z/analyzing-pdf-files-a-deceitful-malware-specie-68eba7b8d086 (accessed 5.7.23).

Cuckoo Sandbox - Automated Malware Analysis [WWW Document], n.d. URL https://cuckoosandbox.org/download (accessed 5.5.23).

Esparza, J.M., n.d. peepdf: UNKNOWN.

Linux malware analysis tools [WWW Document], n.d. . Linux Secur. Expert. URL https://linuxsecurity.expert/security-tools/linux-malware-analysis-tools (accessed 5.7.23).

MultiScanner review (file scanning and analysis framework) [WWW Document], 2019. . Linux Secur. Expert. URL https://linuxsecurity.expert/tools/multiscanner/ (accessed 5.7.23).

pdfid | Kali Linux Tools [WWW Document], n.d. . Kali Linux. URL https://www.kali.org/tools/pdfid/ (accessed 5.7.23).

REMnux: A Linux Toolkit for Malware Analysis [WWW Document], n.d. URL https://docs.remnux.org/ (accessed 5.7.23).

REMnux Usage Tips for Malware Analysis on Linux [WWW Document], n.d. URL https://zeltser.com/remnux-malware-analysis-tips/ (accessed 5.7.23).

Security, H.N., 2022. 7 open-source malware analysis tools you should try out. Help Net Secur. URL https://www.helpnetsecurity.com/2022/08/23/7-open-source-malware-analysis-tools-you-should-try-out/ (accessed 5.5.23).

Setting Up A Kali Linux VM For Malware Analysis – Systran Box [WWW Document], n.d. URL https://www.systranbox.com/how-to-practice-malware-analysis-on-kali-linux/ (accessed 5.8.23).

Setting up Cuckoo Sandbox Step by Step Guide(Malware Analysis Tool) | by Lahiru Oshara Hinguruduwa | Medium [WWW Document], n.d. URL https://medium.com/@oshara.16/setting-up-cuckoo-sandbox-for-dummies-malware-analysis-3daa99e950b5 (accessed 5.7.23).

VirusTotal - Home [WWW Document], n.d. URL https://www.virustotal.com/gui/home/upload (accessed 5.8.23).