

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

Contents.....	2
Table of Figures.....	3
1 - Introduction	6
1.1 – Scenario	6
1.2 – Environment & Tools.....	6
1.2.1 – Environment	6
1.2.2 - Analysis Tools	6
1.3 – Tasks & Questions	7
1.3.1 – Analysis Questions & Tasks Part 1	7
1.3.2 – Analysis Questions & Tasks Part 2	9
2 – Indexes – Malware Analysis Journey & Evidence	12
2.1 - Static & Dynamic Analysis of Unknown Suspicious Files	12
2.1.1 - Static & Dynamic Analysis of PDF Files	12
2.1.2 – Dynamic and Static Analysis of ‘unknown’ Zipped File	17
3 - Analysis & Reverse Engineering of a Malicious DLL	31
3.1 - Scenario & Goal.....	31
3.2 - Environment & Tools	31
3.3 - Basic Static Analysis of ‘malsample.dll’	32
3.4 – Dynamic Analysis of ‘malsample.dll’	38
3.5 – Network Analysis of ‘malsample.dll’	42
3.6 – Reverse Engineering of ‘malsample.dll’	42
4 – References	48

Table of Figures

Table.1.2.2.1 – Tools used during analysis stage.....	6
Figure.2.1.2.1.1 - PDF Sample 1 Check on VirusTotal, demonstrates no malicious indicators.	12
Figure.2.1.2.1.2 - Sample 2.....	13
VirusTotal Analysis Detects Malware.....	13
Figure.2.1.2.1.3 - Sample 2 VirusTotal Details, Points To Embedded Executable.	13
Figure.2.1.2.1.4 – PDF Sample 1 - PDFiD	13
Figure.2.1.2.1.5 – PDF Sample 2 - PDFiD	14
Figure.2.1.2.1.6 – Sample1 - PeePDF.....	14
Figure.2.1.2.1.7 – Error message for sample2 - PeePDF.....	14
Figure.2.1.2.1.8 – Sample2 – PeePDF output 1.....	15
Figure.2.1.2.1.9 – Sample2 – PeePDF output 2.....	15
Figure.2.1.2.1.10 – PDF Parser Confirming Sample 2 Information	15
Figure.2.1.2.1.11 – WindowsXP Sample1 Wireshark Capture	16
Figure.2.1.2.1.12 – REMnux Sample1 Wireshark Capture	16
Figure.2.1.2.1.13 – Sample 2 Wireshark Capture.....	16
Figure.2.1.3.1.1 – PEview 1	17
Figure.2.1.3.1.2 – PEview 2	18
Figure.2.1.3.1.3 – PEview 3	18
Figure.2.1.3.1.4 - PEview 4 - shows here that there is potentially read, write and execute functionality.	18
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.	19
Figure.2.1.3.1.6 - Sections Viewer of the file, note the UPX1, this suggests it is packed.	19
Figure.2.1.3.1.7 - Strings in PEID, you can see here that there is a url where there shouldn't be.	19
Figure.2.1.3.1.8 - Import table discovered through PEID.....	20
Figure.2.1.3.1.9 - Additional Details, not yet sure if significant.	20
Figure.2.1.3.1.10 - Dump of Import.....	20
Figure.2.1.3.1.11 - Dump of File Header.....	21
Figure.2.1.3.1.12 - Dump of DOS Header.	21
Figure.2.1.3.1.13 - Of unknown.file entry point.....	21
Figure.2.1.3.1.14 – Header, Sections and Imports of 'unknown' sample	21
Figure.2.1.3.1.15 - General Information.	22
Figure.2.1.3.1.16 - Suspicious Hex Info.	22
Figure.2.1.3.1.17(a) - Hashes.....	22
Figure.2.1.3.1.17(b) - Hashes	23
Figure.2.1.3.1.18(a) (x4 images) - Header Sections Information, (MZ, PE).....	23
Figure.2.1.3.1.18(b) (x4 images) - Header Sections Information, (MZ, PE).....	23
Figure.2.1.3.1.18(c) (x4 images) - Header Sections Information, (MZ, PE).....	24
Figure.2.1.3.1.18(d) (x4 images) - Header Sections Information, (MZ, PE).....	24
Figure.2.1.3.1.19 - PE Sections Information & Graph.	24
Figure.2.1.3.1.20 - PE32 Structure Information.....	25
Figure.2.1.3.1.21 - Classification Sources, (Blacklisted).....	25
Figure.2.1.3.1.22 - Header analysis.....	25
Figure.2.1.3.1.23 - Sections Analysis.	25
Figure.2.1.3.1.24 - Directories Analysis.	26
Figure.2.1.3.1.25 - Imports Analysis.	26
Figure.2.1.3.1.26 - Resources Analysis. (Note the url from earlier).....	26

Figure.2.1.3.1.27 - Strings Analysis. (There are 249 strings, note from 224 to 249).....	26
Figure.2.1.3.1.28 - Load Config Analysis.	27
Figure.2.1.3.1.29 - Hex View Analysis. (Note it is the same place as the strings)	27
Figure.2.1.3.2.1 – Unpacked ‘unknown’ sample.....	27
Figure.2.1.3.2.2 - PEiD Now shows Microsoft Visual C++ 8* and differing information.	28
Figure.2.1.3.2.3 - PEview now has more information.	28
Figure.2.1.3.2.4 - PEview Config Directory.....	28
Figure.2.1.3.2.5 - PEview DLL Names.	29
Figure.2.1.3.2.6 - The architecture is now different in PEview.....	29
Figure.2.1.3.2.7 – VirusTotal output	29
Figure.2.1.3.2.8 - Image of change of file to .exe, note the logo and type change.	30
Figure.2.1.3.2.9 – RegShot Comparison	30
Figure.2.1.3.3.1 – Wireshark Capture for ‘unknown’ sample.....	30
Figure.2.1.3.3.2 – FakeNet Capture for ‘unknown’ sample	31
Figure.2.1.3.3.3 – Procmon Capture for ‘unknown’ sample	31
Table.3.2.1 – Tools used for ‘malsample.dll’	31
Figure.3.3.1 – MiTec EXE Sections Tab Information	32
Figure.3.3.2 – MiTec EXE Strings Tab Information.....	32
Figure.3.3.3 – MiTec EXE Header Tab Information	33
Figure.3.3.3 – PEiD Information.....	33
Figure.3.3.4 - PEiD System Tasks Information	33
Figure.3.3.5 - FileAlyzer General Tab Information	34
Figure.3.3.6 - FileAlyzer Hex Tab.....	34
Figure.3.3.7 - FileAlyzer OpenSBI.....	34
Figure.3.3.8 - FileAlyzer Security	35
Figure.3.3.9 - FileAlyzer PE Header Tab.....	35
Figure.3.3.10(a) - FileAlyzer PE Sections Tab	35
Figure.3.3.10(b) - FileAlyzer PE Sections Graph Tab	35
Figure.3.3.10(c) - FileAlyzer PE Sections Imports Tab (ADVAPI(L) & KERNEL32(R)).....	36
Figure.3.3.10(d) - FileAlyzer PE Sections Exports Tab	36
Figure.3.3.11(a) - FileAlyzer PE Imports Tab ADVAPI32.dll	36
Figure.3.3.11(b) - FileAlyzer PE Imports Tab KERNELL32.dll	37
Figure.3.3.12 - FileAlyzer PE Exports Tab	37
Figure.3.3.13 - FileAlyzer Disassembler Tab	37
Figure.3.3.14 - FileAlyzer Classification Tab.....	38
Figure.3.4.1 – Unpacked malsample	38
Figure.3.4.2 - RegShot Capture of ‘malsample.dll’ 1.....	38
Figure.3.4.3 – Command Line of ‘malsample.dll’	39
Figure.3.4.5– FakeNet Monitor Output.....	39
Figure.3.4.6 – dll run command	39
Figure.3.4.7 – Process Explore Prior to Execution	40
Figure.3.4.8 – Process Explore After to Execution	40
Figure.3.4.9 – Process Explore Internet Explorer Properties, then Strings tab	41
Figure.3.4.10 – Process Monitor Filtered to rundll32.exe	41
Figure.3.5.1 – Wireshark after installing service (1).....	42
Figure.3.5.2 – Wireshark after installing service (2).....	42
Figure.3.6.1 – malsample.dll Exports and Addresses (IDA).....	42
Figure.3.6.2 – kernell32.dll LoadLibrary (IDA)	43

Figure.3.6.3 – kernel32 API Sleep() (IDA)	43
Figure.3.6.4 – Graph View of ServiceMain (IDA)	43
Figure.3.6.4(a) – Answer to question 5b	44
Figure.3.6.5 – Exports (IDA)	44
Figure.3.6.6 – Functions (IDA)	44
Figure.3.6.7 – Strings (IDA)	45
Figure.3.6.8 – Strings 2 (IDA)	46
Figure.3.6.9 – Strings 3 (IDA)	46
Figure.3.6.10 - Structures (IDA)	47
Figure. 3.6.11– Enums (IDA).....	47
Figure.3.6.12 – Imports (IDA).....	47

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 I 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	dc3f10f2d8123ea1317c716a028cd2f96b3d982243ae6564d1ef3c51976a85c
Vhash	9dd3f7ed297090f55e8a830b65bbf9969
SSDEEP	3072:FE+3ltdFB6rK8+KQotdTgKr/jjutiDm4Z6MVT8TxS: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.

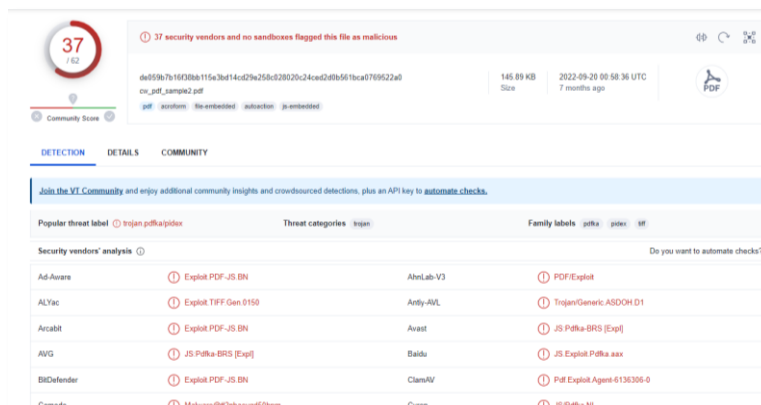


Figure.2.1.2.1.2 - Sample 2

VirusTotal Analysis Detects Malware

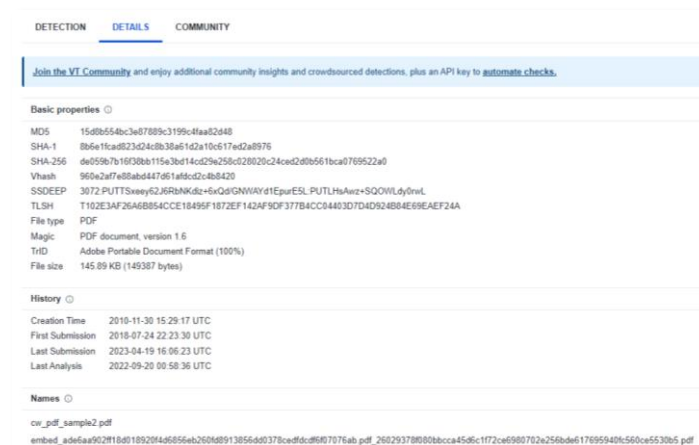


Figure.2.1.2.1.3 - Sample 2 VirusTotal Details, Points To Embedded Executable.

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.

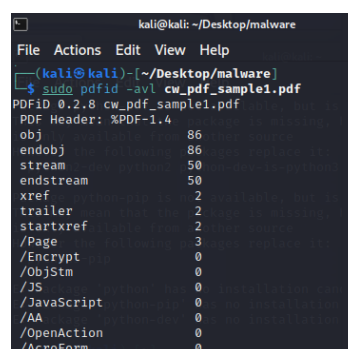


Figure.2.1.2.1.4 – PDF Sample 1 - PDFiD

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.

```
PDFiD 0.2.8 cw_pdf_sample2.pdf
PDF Header: %PDF-1.6
obj          146
endobj       146
stream       55
endstream    55
xref         1
trailer      1
startxref   1
/Page        1
/Encrypt     0
/ObjStm      0
/JS          2
/JavaScript   2
/AA          2
/OpenAction  0
/AcroForm    1
/JBIG2Decode 0
/RichMedia   0
/Launch      0
/EmbeddedFile 1
```

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.

```
File: cw_pdf_sample1.pdf
MD5: c30c96c9d9e9bf9a454ab0b8fb754b14
SHA1: 05433eccfealc825b760a415d1ef432eadfb0c74
SHA256: dc3f10f2d8123ea1317c716a028cd2ff96b3d982243ae6564d1ef3c51976a85c
Size: 139996 bytes
Version: 1.4
Binary: True
Linearized: True
Encrypted: False
Updates: 1
Objects: 86
Streams: 50
URIs: 0
Comments: 0
Errors: 0

Version 0:
  Catalog: 26
  Info: 24
  Objects (1): [25]
  Streams (0): []

Version 1:
  Catalog: No
  Info: No
  Objects (85): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86]
  Streams (50): [86, 40, 42, 43, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 66, 67, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79,
```

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.

```
remnux@remnux:~/Downloads$ peepdf cw_pdf_sample2.pdf
Error: An error has occurred while parsing an indirect object!!
remnux@remnux:~/Downloads$
```

Figure.2.1.2.1.7 – Error message for sample2 - PeePDF.

```

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

File: cw_pdf_sample2.pdf
MD5: 15d8b554bc3e87889c3199c4faa82d48
SHA1: 8b6e1fcad823d24c8b38a61d2a10c617ed2a8976
SHA256: de059b7b16f38bb115e3bd14cd29e258c028020c24ced2d0b561bca0769522a0
Size: 149387 bytes
Version: 1.6
Binary: False
Linearized: True
Encrypted: False
Updates: 0
Objects: 146
Streams: 55
RIS: 0
Comments: 0
Errors: 0

Version 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, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 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, 121, 123, 126, 128, 131, 133, 136, 138, 140, 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, 93, 96, 98, 101, 103, 106, 108, 111, 113, 116, 118, 121, 123, 126, 128, 131, 133, 136, 138, 141, 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

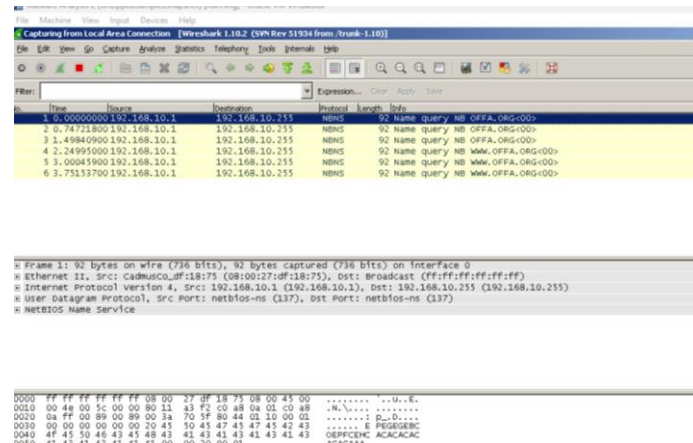


Figure.2.1.2.1.11 – WindowsXP Sample1 Wireshark Capture

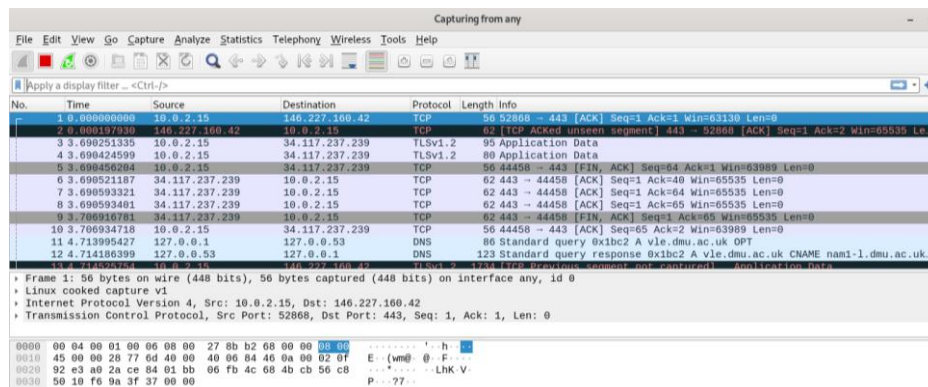


Figure.2.1.2.1.12 – REMnux Sample1 Wireshark Capture

Sample 2:

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

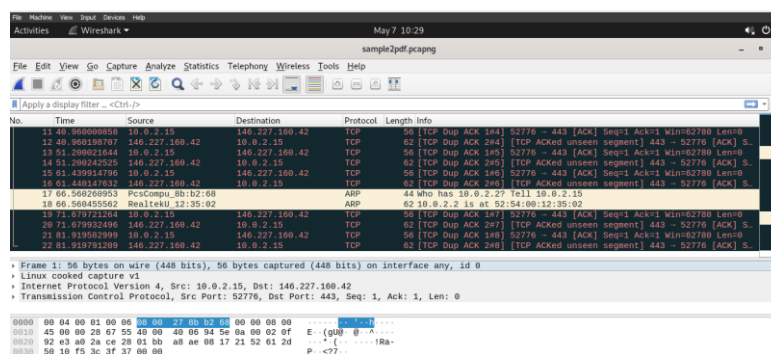


Figure.2.1.2.1.13 – Sample 2 Wireshark Capture

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.

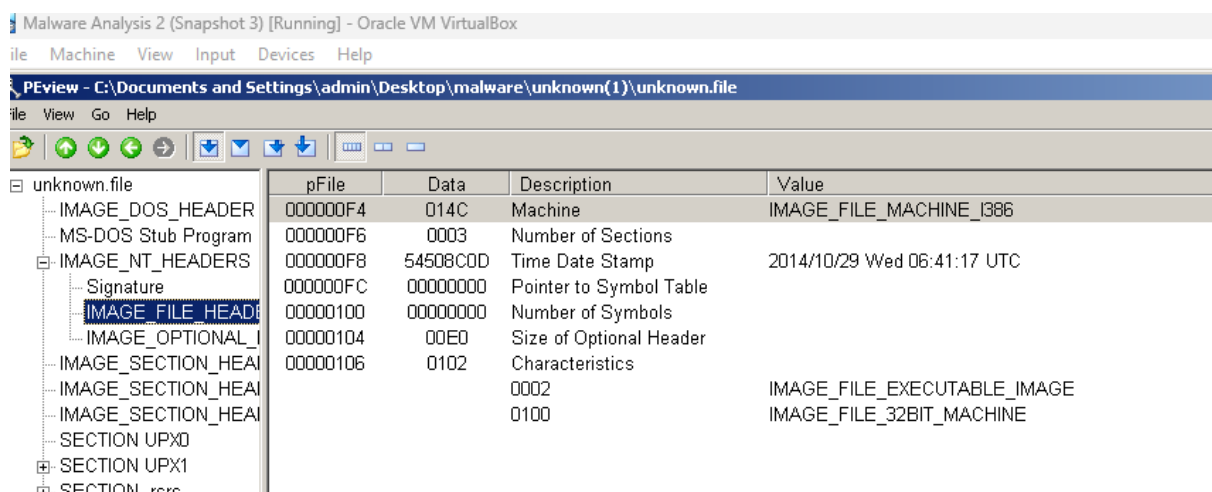
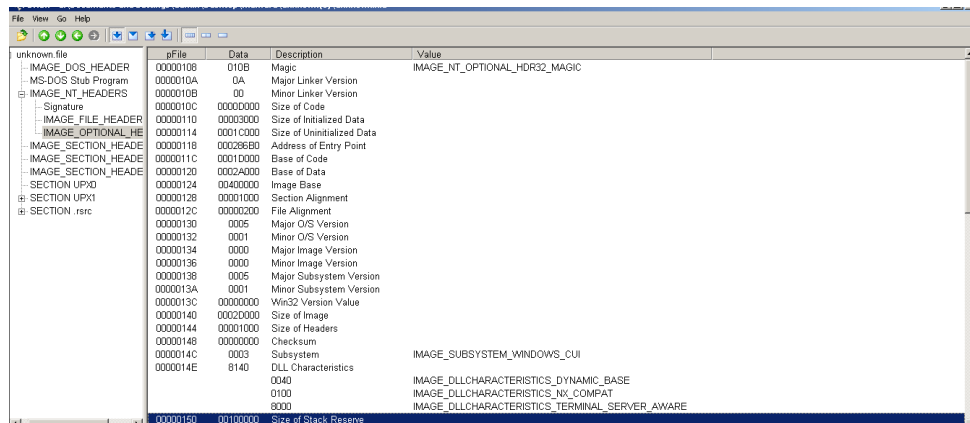
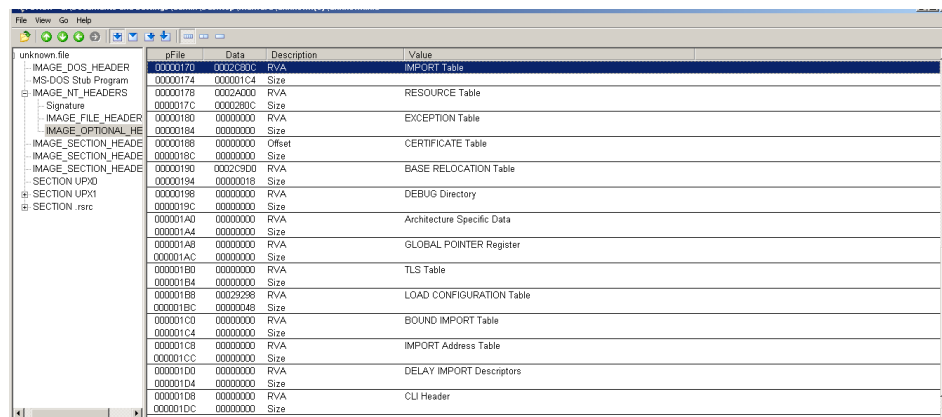


Figure.2.1.3.1.1 – PEview 1



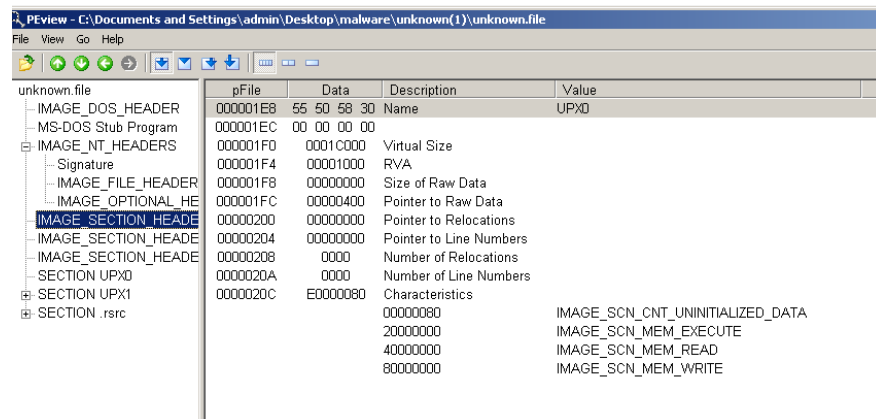
pFile	Data	Description	Value
00000108	010B	Magic	IMAGE_NT_OPTIONAL_HDR32_MAGIC
0000010A	0A	Major Linker Version	
0000010B	00	Minor Linker Version	
0000010C	00000000	Size of Code	
00000110	00003000	Size of Initialized Data	
00000114	0001C000	Size of Uninitialized Data	
00000118	000296B0	Address of Entry Point	
0000011C	00010000	Base of Code	
00000120	0002A000	Base of Data	
00000124	00400000	Image Base	
00000128	00001000	Section Alignment	
0000012C	00000200	File Alignment	
00000130	0005	Major OS Version	
00000132	0001	Minor OS Version	
00000134	0000	Major Image Version	
00000136	0000	Minor Image Version	
00000138	0005	Major Subsystem Version	
0000013A	0001	Minor Subsystem Version	
0000013C	00000000	Win32 Version Value	
00000140	00020000	Size of Image	
00000144	00001000	Size of Headers	
00000148	00000000	Checksum	
0000014C	0003	Subsystem	IMAGE_SUBSYSTEM_WINDOWS_CUI
0000014E	8140	DLL Characteristics	0040 0100 8000
00000150	00100000	Size of Stack Reserve	

Figure.2.1.3.1.2 – PEview 2



pFile	Data	Description	Value
00000170	0002C00C	RVA	IMPORT Table
00000174	000001C4	Size	
00000178	00024000	RVA	RESOURCE Table
0000017C	0000280C	Size	
00000180	00000000	RVA	EXCEPTION Table
00000184	00000000	Size	
00000188	00000000	Offset	CERTIFICATE Table
0000018C	00000000	Size	
00000190	0002C350	RVA	BASE RELOCATION Table
00000194	00000018	Size	
00000198	00000000	RVA	DEBUG Directory
0000019C	00000000	Size	
000001A0	00000000	RVA	Architecture Specific Data
000001A4	00000000	Size	
000001A8	00000000	RVA	GLOBAL POINTER Register
000001AC	00000000	Size	
000001B0	00000000	RVA	TLS Table
000001B4	00000000	Size	
000001B8	0002C298	RVA	LOAD CONFIGURATION Table
000001BC	00000048	Size	
000001C0	00000000	RVA	BOUND IMPORT Table
000001C4	00000000	Size	
000001C8	00000000	RVA	IMPORT Address Table
000001CC	00000000	Size	
000001D0	00000000	RVA	DELAY IMPORT Descriptors
000001D4	00000000	Size	
000001D8	00000000	RVA	CLI Header
000001DC	00000000	Size	

Figure.2.1.3.1.3 – PEview 3



pFile	Data	Description	Value
000001E8	55 50 58 30	Name	UPX0
000001EC	00 00 00 00		
000001F0	0001C000	Virtual Size	
000001F4	00001000	RVA	
000001F8	00000000	Size of Raw Data	
000001FC	00000400	Pointer to Raw Data	
00000200	00000000	Pointer to Relocations	
00000204	00000000	Pointer to Line Numbers	
00000208	0000	Number of Relocations	
0000020A	0000	Number of Line Numbers	
0000020C	E0000080	Characteristics	
00000080		IMAGE_SCN_CNT_UNINITIALIZED_DATA	
20000000		IMAGE_SCN_MEM_EXECUTE	
40000000		IMAGE_SCN_MEM_READ	
80000000		IMAGE_SCN_MEM_WRITE	

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

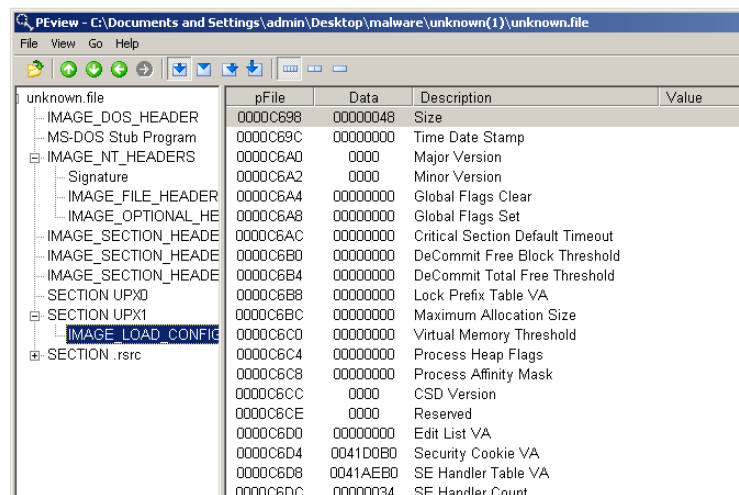


Figure.2.1.3.1.5 - PEView 5 - Here you can see the raw data and some words 'access, privileges, security' indicating this is definitely not an image.

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.

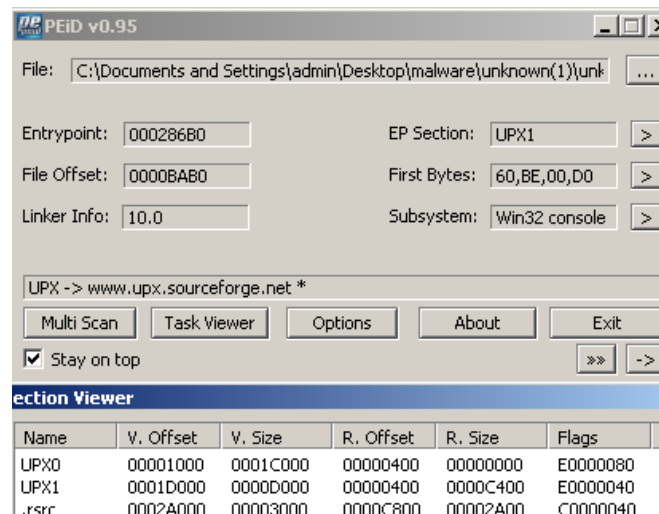


Figure.2.1.3.1.6 - Sections Viewer of the file, note the UPX1, this suggests it is packed.

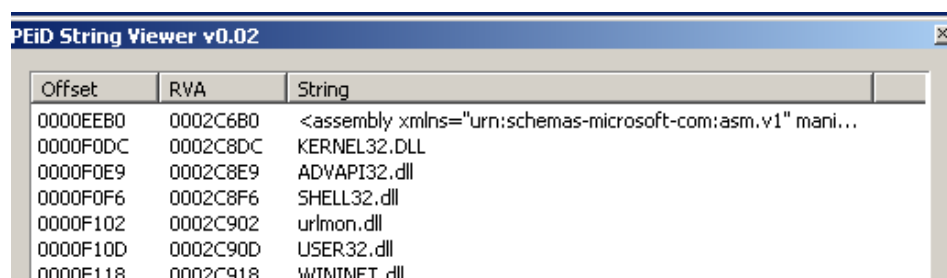


Figure.2.1.3.1.7 - Strings in PEiD, you can see here that there is a url where there shouldn't be.

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.

Imports Viewer						
DllName	OriginalFirstThunk	TimeDateStamp	ForwarderChain	Name	FirstThunk	
KERNEL32.DLL	00000000	00000000	00000000	0002C8DC	0002C898	
ADVAPI32.dll	00000000	00000000	00000000	0002C8E9	0002C8B4	
SHELL32.dll	00000000	00000000	00000000	0002C8F6	0002C8BC	
urlmon.dll	00000000	00000000	00000000	0002C902	0002C8C4	
USER32.dll	00000000	00000000	00000000	0002C90D	0002C8CC	
WININET.dll	00000000	00000000	00000000	0002C918	0002C8D4	

Figure.2.1.3.1.8 - Import table discovered through PEID.

PE Details	
Basic Information	
EntryPoint: 000286B0	SubSystem: 0003
ImageBase: 00400000	NumberOfSections: 0003
SizeOfImage: 0002D000	TimeDateStamp: 54508C0D
BaseOfCode: 0001D000	SizeOfHeaders: 00001000
BaseOfData: 0002A000	Characteristics: 0102
SectionAlignment: 00001000	Checksum: 00000000
FileAlignment: 00000200	SizeOfOptionalHeader: 00E0
Magic: 010B	NumOfRvaAndSizes: 00000010
Directory Information	
ExportTable: 00000000	RVA: 00000000 SIZE: 00000000
ImportTable: 0002C80C	RVA: 000001C4 SIZE: 000001C4
Resource: 0002A000	RVA: 0000280C SIZE: 0000280C
TLSTable: 00000000	RVA: 00000000 SIZE: 00000000
Debug: 00000000	RVA: 00000000 SIZE: 00000000
Close	

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.

Dump of Import																
	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B
0x0042C87C	18	C9	02	00	D4	C8	02	00	00	00	00	00	00	00	00	00
0x0042C88C	00	00	00	00	00	00	00	00	00	00	00	00	24	C9	02	00
0x0042C89C	32	C9	02	00	42	C9	02	00	52	C9	02	00	60	C9	02	00
0x0042C8AC	6E	C9	02	00	00	00	00	00	7C	C9	02	00	00	00	00	00
0x0042C8BC	8A	C9	02	00	00	00	00	00	9A	C9	02	00	00	00	00	00
0x0042C8CC	AE	C9	02	00	00	00	00	00	BA	C9	02	00	00	00	00	00
0x0042C8DC	4B	45	52	4E	45	4C	33	32	2E	44	4C	4C	00	41	44	56
0x0042C8EC	41	50	49	33	32	2E	64	6C	6C	00	53	48	45	4C	4C	33
0x0042C8FC	32	2E	64	6C	6C	00	75	72	6C	6D	6F	6E	2E	64	6C	6C
0x0042C90C	00	55	53	45	52	33	32	2E	64	6C	6C	00	57	49	4E	49
0x0042C91C	4E	45	54	2E	64	6C	6C	00	00	00	4C	6F	61	64	4C	69
0x0042C92C	62	72	61	72	79	41	00	00	47	65	74	50	72	6F	63	41
0x0042C93C	64	64	72	65	73	73	00	00	56	69	72	74	75	61	6C	50
0x0042C94C	72	6F	74	65	63	74	00	00	56	69	72	74	75	61	6C	41
0x0042C95C	6C	6C	6F	63	00	00	56	69	72	74	75	61	6C	46	72	65
0x0042C96C	65	00	00	00	45	78	69	74	50	72	6F	63	65	73	73	00
0x0042C97C	00	00	52	65	67	43	6C	6F	73	65	4B	65	79	00	00	00
0x0042C98C	53	68	65	6C	6C	45	78	65	63	75	74	65	41	00	00	00
0x0042C99C	55	52	4C	44	6F	77	6E	6C	6F	61	64	54	6F	46	69	6C
0x0042C9AC	65	41	00	00	53	68	6F	77	57	69	6E	64	6F	77	00	00
0x0042C9BC	44	65	6C	65	74	65	55	72	6C	43	61	63	68	65	45	6E
0x0042C9CC	74	72	79	00												

Figure.2.1.3.1.10 - Dump of Import.

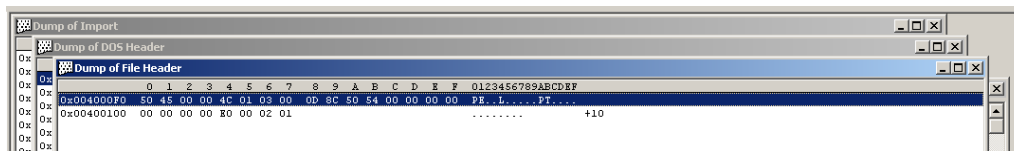


Figure.2.1.3.1.11 - Dump of File Header.

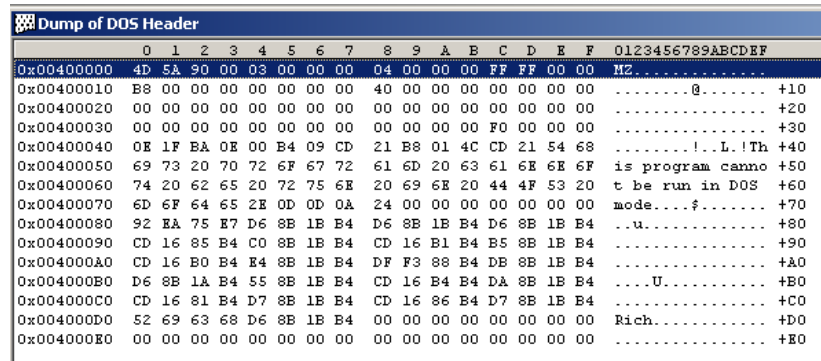


Figure.2.1.3.1.12 - Dump of DOS Header.

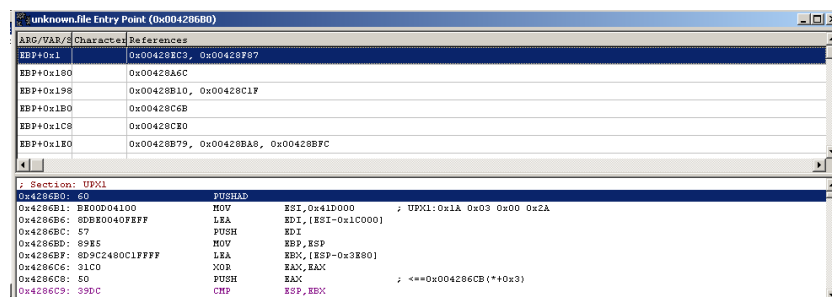


Figure.2.1.3.1.13 - Of unknown.file entry point.

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

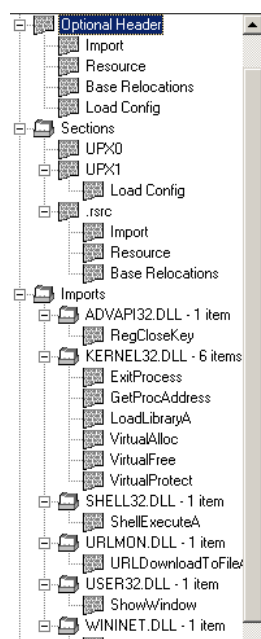


Figure.2.1.3.1.14 – Header, Sections and Imports of 'unknown' sample

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.

FileAlyzer 2.0.5.57 - [C:\Documents and Settings\admin\Desktop\malware\unknown(1)\unknown.file]	
File Edit Window Report Help	
General Hex Anomalies OpenSBI Map Bitmap Streams Security Hashes MZ Header PE Header PE Sections PE Imports PE Exports PE Resources Disassembler Compatibility Classification Sources VirusTotal	
unknown.file	
Location: C:\Documents and Settings\admin\Desktop\malware\unknown(1)\	
Size: 136704 0000000000021600	
Version:	
RC-32: 38A14896	
D5: F0E4CF37573CFAC1E46E6F1FCDAC7C10	
SHA-1: 85FDA3811FBAD541A7DF8A050CD4847A55D4D3	
Read only: <input type="checkbox"/> Directory <input type="checkbox"/> Archive <input type="checkbox"/> Symbolic link	
Hidden: <input type="checkbox"/> System file: <input type="checkbox"/>	
Creation: Saturday, April 22, 2023 5:03:28 PM 2023-04-22 17:03:28	
Last access: Saturday, April 22, 2023 7:52:10 PM 2023-04-22 19:52:10	
Last write: Tuesday, March 23, 2021 8:33:12 PM 2021-03-23 20:33:12	
Creation (UTC): Saturday, April 22, 2023 3:03:28 PM 2023-04-22 15:03:28	
Last access (UTC): Saturday, April 22, 2023 5:52:10 PM 2023-04-22 17:52:10	
Last write (UTC): Tuesday, March 23, 2021 6:33:12 PM 2021-03-23 18:33:12	

Figure.2.1.3.1.15 - General Information.

Serial	Hex	Anomalies	OpenSBI	Map	Bitmap	Streams	Security	Hashes	MZ Header	PE Header	PE Sections	PE Imports	PE Exports	PE Resources	Disassembler	Compatibility	Classification Sources		
0x1B1BC	7572	6365	4578	5700	0000	4765	7443	6F6E	736F	6C65	4D6F	6465	0000	4765	7443	6F6E	736F	6C65	4350
0x1B1B2	0000	5274	6C55	6377	6968	6400	0000	466C	7573	6846	696C	6542	7566	6665	7273	0000	4765	7453	7472
0x1B208	6968	6754	7970	6557	0000	4765	7443	7572	7265	6874	5072	6F63	6573	7349	6400	0000	5261	6973	6545
0x1B22E	7963	6570	7469	6968	0000	4568	7465	7243	7269	7469	6361	6C53	6563	7469	6F68	0000	4C56	6176	6543
0x1B254	7269	7469	6361	6C53	6563	7469	6F68	0000	4765	744C	6173	7445	7272	6972	0000	496E	6974	6961	6C69
0x1B27A	7465	4372	6974	6963	616C	5365	6374	696F	6841	6864	5370	6968	436F	756E	7400	0000	4465	6C65	7465
0x1B2A0	4372	6974	6963	616C	5365	6374	696F	6800	0000	4865	6170	4465	7374	726F	7900	0000	4865	6170	416C
0x1B2C6	6C6F	6300	0000	4865	6170	4672	6565	0000	4865	6170	5265	416C	6C6F	6300	0000	4865	6170	5369	7465
0x1B2EC	0000	4765	7450	726F	6365	7373	4865	6170	0000	4765	7453	7973	7465	6D54	696D	6541	7346	696C	6554
0x1B312	696D	6500	0000	4765	7443	6F6D	6D61	6864	4C69	6865	4100	0000	4865	6170	5365	7449	6866	6772	6D61
0x1B338	7469	6F6E	0000	4568	636F	6465	506F	6968	7465	7200	0000	4465	636F	6465	506F	696E	7465	7200	0000
0x1B35E	556E	6861	6864	6C65	6445	7963	6570	7469	6F68	4669	6C74	6572	0000	5365	7455	6868	616E	646C	6564
0x1B384	4578	6365	7074	696F	6846	696C	7465	7200	0000	4973	4465	6275	6767	6572	5072	6573	656E	7400	0000
0x1B3AA	4765	7443	5049	6866	6F00	0000	4968	7465	726C	6F63	6865	6449	6863	7265	6D65	6874	0000	496E	7465
0x1B3D0	726C	6F63	6865	6444	6563	7265	6D65	6874	0000	4765	7441	4350	0000	4765	744F	454D	4350	0000	4973
0x1B3F6	5661	6C69	6443	6F64	6550	6167	6500	0000	546C	7341	6C6C	6F63	0000	546C	7347	6574	5661	6C75	6500
0x1B41C	0000	546C	7353	6574	5661	6C75	6500	0000	546C	7346	7265	6500	0000	4765	744D	6F64	756C	6548	616E
0x1B442	646C	6557	0000	5365	744C	6173	7445	7272	6972	0000	4765	7443	7572	7265	6874	6468	7265	6164	4964
0x1B468	0000	4C43	4D61	7053	7472	6968	6757	0000	4D75	6C74	6942	7974	6554	6875	6964	6543	6861	7200	0000
0x1B496	4973	5072	6F63	6573	736F	7246	6561	7475	7265	6573	6568	7400	0000	4578	6974	5072	6F63	6573	
0x1B4B4	7300	0000	4765	744D	6F64	756C	6546	696C	6548	616D	6557	0000	4672	6565	456E	7669	726F	686D	656E
0x1B4DA	7453	7472	6968	6773	5700	0000	4765	7445	6876	6972	6F68	6D65	6874	5374	7269	6867	7357	0000	5365
0x1B500	7448	616E	646C	6543	6775	6874	0000	4765	7446	696C	6554	7970	6500	0000	4765	7453	7461	7274	7570
0x1B526	4968	666F	5700	0000	4865	6170	4372	6561	7465	0000	5175	6572	7950	6572	666F	726D	616E	6365	436F
0x1B54C	756E	7465	7200	0000	4765	7454	6963	6843	6F75	6874	0000	5265	6753	6574	5661	6C75	6545	7841	0000
0x1B572	5265	6745	6875	6D4B	6579	4578	4100	0000	5265	6743	6C6F	7365	4865	7900	0000	5265	674F	7065	684B
0x1B598	6579	4578	4100	0000	5265	6751	7565	7279	5661	6C75	6545	7841	0000	5265	6743	7265	6174	654B	6579
0x1B5BE	4578	4100	0000	5368	6565	6C45	7965	6375	7465	4578	4100	0000	5348	4765	7453	7065	6369	616C	466F

Figure.2.1.3.1.16 - Suspicious Hex Info.

Cyclic redundancy check	
CRC-32: Cyclic redund...	38A14896
Message-Digest algorithm	
MD2: Message-Digest ...	49381F9D3A1D435C9908E0F9C683CCF6
MD4: Message-Digest ...	A187E404125152A8ECC30F71D172562E
e2dk:	5C0135bc1f0a1d5d8375ab71d9d6298d
MD5: Message-Digest ...	F0E4CF37573CFAC1E46E6F1FCDAC7C10
US Secure Hash Algorithm	
SHA-0: US Secure Has...	6EA27994D2C885A471801D7EBAC02476D5C9DF3E
SHA-1: US Secure Has...	85FDA3811FBAD541A7DF8A050CD4847A55D4D3
SHA-256: US Secure H...	EB9048578FED559818062B7C3A013117609E8D743D06F391FE11BD13DEA9C6A9
SHA-384: US Secure H...	3E494A36881F870EE34B2FF7E088FA099817F2ABBD250FDD93AFCE99C81812735CCEACAF5D4BE43E02886E6A778DAC58
SHA-512: US Secure H...	AA09E84ADD9ECBAAD111103F9A0D6D840B836212A344462421CECC6D20FB26F6D9AFEC7B5464357F5FBA036412E38F6373BDDC45099A66BE9F52C101382E3
RAPE Integrity Primitives Evaluation MD	
RipeMD-128: RACE In...	CF0A3A7363D2994B6426C1DCB245A2D5
RipeMD-160: RACE In...	800434B100A62D8B2598962BEF127E2EB909D5B6
RipeMD-256: RACE In...	C5ACF1BF907547D8A223872700D4AC31D6080D8949DF2CA8CB87B038E24B4D54
RipeMD-320: RACE In...	70E7761E976541AC8F90BCD6CF2977EECC263DE8C8B09E380C8B8C49DC798F9E9C4504872CA7FD8
HAVAL	
HAVAL-128	075CEAD0B9D5192692FD274AAB30B474

Figure.2.1.3.1.17(a) - Hashes.

```

RipeMD-160: RACE In... B00434B100A62DB82598962BEF127E2EB909D586
RipeMD-256: RACE In... C5ACF1BF907547D8A223B722700D4AC31D6080DB849DF2CA8CB7B038E24B4D54
RipeMD-320: RACE In... 70E77616E976541AC8F90BCB06CF297EEC263DE3C8B09E380C8B8C49DC79BF9E9C4504872CA7FD8

HAVAL

HAVAL-128 075CEAD0B9D5192692FD274AAB30B474
HAVAL-160 483AF7EE68D7DF40523F18E408031BF842EDF3C0
HAVAL-192 89A1977E152577E680CE036C569082D192852CCCE013665DF
HAVAL-224 981889CBE30394B4D3E2D51862B1BC98C676D60676C8B316D796A25E
HAVAL-256 86A6C8FF1040A68A41D0428FC21DF76F1F89672EFDAD96F52FA284135ECC747B

Other

Snefru-128 F7D46433738EE9896F64157237817DA5
Snefru-256 BA045017696EF4D08728F9478C21AAA1B8904C139F0F7110A72D857F195180C3
Tiger-192 873F496DCD86926D9521BBD059F7E24EFCDC71F8235367DC
Panama 48BE6D7B5D9A338F055CB565DD58D4E688AF5134D27D0EF279F84A07C3AB76F3
Square 586AC4E81678C3D8FEA94A6F3EFD0C613
SSDeap 1536:WTLb50hCIW7SrxenJYx6Mn1gaQq3d6YuAOXNHb+w6h5SyDyz+0cLQoG6fYckT5:CbSABGF5YuAKkwsyD5f7gX/jwqAb

```

Figure.2.1.3.1.17(b) - Hashes.

General Hex Anomalies OpenSBI Map Bitmap Streams Security Hashes MZ Header PE Header PE Sections PE	
<input checked="" type="checkbox"/> Highlight partial results	
Field	Data
MZ header	
Signature	5A4D
Last Page Size	0000
Total Pages In File	0003
Relocation Items	0000
Paragraphs	0004
MZ DOS header	
Min Extra Paragraphs	0000
Max Extra Paragraphs	FFFF
Initial Stack Segment	0000
Initial Stack Pointer	0000
Checksum for Header	0000
Initial Instruction Pointer	0000
Initial Code Segment	0000
Relocation Table Offset	0040
Overlay Number	00000
Reserved #0	00000000
Reserved #1	00000000
Reserved #2	00000000
Reserved #3	00000000
Reserved #4	00000000
Reserved #5	00000000
Reserved #6	00000000
Reserved #7	00000000
PE Header Pointer	00000FD0

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

General | Hex | Anomalies | OpenSBI | Map | Bitmap | Streams | Security | Hashes | MZ Header | PE Header

☒ Highlight partial results

Header | Graph

Field	Data	Details
PE header		
Signature	00004550	
Machine	014C	Intel 386
Number of sections	0005	
Time/Date stamp (local)	54508C0D	2014-10-29 08:41:17
Time/Date stamp (UTC)	54508C0D	2014-10-29 06:41:17
Pointer to symbol table	00000000	
Number of symbols	00000000	
Size of optional header	00E0	
Characteristics	0102	Executable, 32bit Machine Expected
PE32 optional header		
Magic	010B	
Version of Linker (major)	0A	
Version of Linker (minor)	00	
Size of code	00015600	
Size of initialized data	0000BC00	
Size of uninitialized data	00000000	
Address of entry point	0000B7B7	
Base of code	00001000	
Base of data	00017000	
Image base	00400000	
Section alignment	00001000	
File alignment	00000200	

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
Copyright Directory Address	00000000
Copyright 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 Address	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).

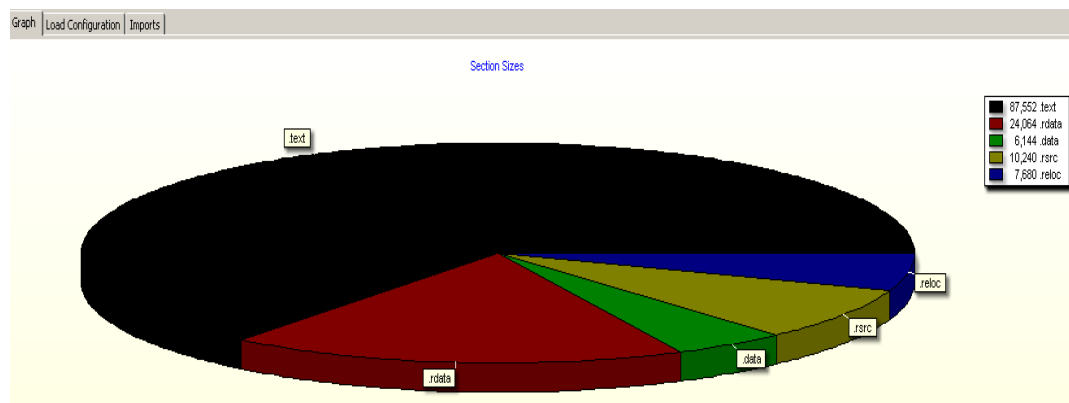


Figure.2.1.3.1.19 - PE Sections Information & Graph.

Field	Data	Details
PE32 Structure		
Characteristics	00000048	
Timestamp	00000000	
Version (major)	0000	
Version (minor)	0000	
Global Flags Clear	00000000	
Global Flags Set	00000000	
Critical Section Default TimeOut	00000000	
DeCommit Free Block Threshold	00000000	
DeCommit Total Free Threshold	00000000	
Lock Prefix Table	00000000	
Maximum Allocation Size	00000000	
Virtual Memory Threshold	00000000	
Process Affinity Mask	00000000	
Process Heap Flags	00000000	
CSD Version	0000	
Edit List	00000000	
Security Cookie	0041D080	
SE Handler Table	0041AE80	
SE Handler Count	00000034	

Figure.2.1.3.1.20 - PE32 Structure Information.

General | Hex | Anomalies | OpenSBI | Map | Bitmap | Streams | Security | Hashes | MZ Header | PE Header | PE Sect

☒ Highlight partial results

Summary | Google | Bing | Yahoo | Threat Expert | Virus Total | Comodo

Service	Status	Timestamp
Blacklists		
Malware Hash Registry (Team Cymru)	n/a	n/a
Comodo	unlisted	2023-04-22 20:12:21
Mixed lists		
Google	unlisted	2023-04-22 20:12:21
Bing	unlisted	2023-04-22 20:12:21
Yahoo	unlisted	2023-04-22 20:12:21

Figure.2.1.3.1.21 - Classification Sources, (Blacklisted).

MiTecEXE

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

MiTeC EXE Explorer - [C:\Documents and Settings\admin\Desktop\malware\unknown(1)\	
File	
Portable Executable - PE32	
32-bit Intel - Console	
Header	Sections
Property	Value
Signature	0x00004550 (Portable Executable)
Machine	32-bit Intel
Number of sections	3
Timestamp	10/29/2014 6:41:17 AM
Pointer to symbol table	0x00000000
Number of symbols	0
Size of optional header	224
Characteristics	0x0102

Figure.2.1.3.1.22 - Header analysis.

Header	Sections	Directories	Imports	Resources	Strings	Load Config	Hex View
Name	Virtual Address	Virtual Size	Raw Data Offset	Raw Data Size	Flags		
UPX0	00001000	114688	00000400	0	E0000080		
UPX1	0001D000	53248	00000400	50176	E0000040		
.rsrc	0002A000	12288	0000C800	10752	C0000040		

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.

Header	Sections	Directories	Imports	Resources
ADVAPI32.dll (1)				
KERNEL32.DLL (6)				
SHELL32.dll (1)				
urlmon.dll (1)				
USER32.dll (1)				
WININET.dll (1)				

Figure.2.1.3.1.25 - Imports Analysis.

Header	Sections	Directories	Imports	Resources	Strings	Load Config	Hex View
ICON							
GROUP_ICON							
MANIFEST							
Property							
Name							
Type							
Language							
Code page							
RVA							
Offset							
Size							

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
00000000	3C	61	73	73	65	6D	62	6C	79	20	78	6D	6C	6E	73	3D	<assembly xmlns=
00000010	22	75	72	68	3A	73	63	68	65	6D	61	73	2D	6D	69	63	"urn:schemas-mic
00000020	72	6F	73	6F	66	74	2D	63	6F	6D	3A	61	73	6D	2E	76	rosoft-com:asm.v
00000030	31	22	20	6D	61	6E	69	66	65	73	74	56	65	72	73	69	l" manifestVersi
00000040	6F	6E	3D	22	31	2E	30	22	3E	0D	0A	20	20	3C	74	72	on="1.0">... <tr
00000050	75	73	74	49	6E	66	6F	20	78	6D	6C	6E	73	3D	22	75	ustInfo xmlns="u
00000060	72	6E	3A	73	63	68	65	6D	61	73	2D	6D	69	63	72	6F	rn:schemas-micro
00000070	73	6F	66	74	2D	63	6F	6D	3A	61	73	6D	2E	76	33	22	soft-com:asm.v3"
00000080	3E	0D	0A	20	20	20	20	3C	73	65	63	75	72	69	74	79	>... <security
00000090	3E	0D	0A	20	20	20	20	3C	72	65	71	75	65	73			>... <reques
000000A0	74	65	64	5A	72	69	76	6A	6F	65	67	65	73	3E	0D	0A	redPrivileges>

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

Header	Sections	Directories	Imports	Resources	Strings	Load Config	Hex View
218	t\$\\tY						
219	91\$\\w_						
220	D\$tIt						
221	91\$tr						
222	[[^_]						
223	XPTPSW						
224	<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">						
225	<trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">						
226	<security>						
227	<requestedPrivileges>						
228	<requestedExecutionLevel level="asInvoker" uiAccess="false"></requestedExecutionLevel>						
229	</requestedPrivileges>						
230	</security>						
231	</trustInfo>						
232	</assembly>PA						
233	KERNEL32.DLL						
234	ADVAPI32.dll						
235	SHELL32.dll						
236	urlmon.dll						
237	USER32.dll						
238	WININET.dll						
239	LoadLibraryA						
240	GetProcAddress						
241	VirtualProtect						
242	VirtualAlloc						
243	VirtualFree						
244	ExitProcess						
245	RegCloseKey						
246	ShellExecuteA						
247	URLDownloadToFileA						
248	ShowWindow						
249	DeleteUrlCacheEntry						

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	ESC8 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
0x0000F0A0	42C9 0200 52C9 0200	60C9 0200	6EC9 0200
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
0x0000F0E0	454C 3332 2E44 4C4C	0041 4456	4150 4933
0x0000F0F0	322E 646C 6C00 5348	454C 4C33	322E 646C
0x0000F100	6C00 7572 6C6D 6F6E	2E64 6C6C	0055 5345
0x0000F110	5233 322E 646C 6C00	5749 4E49	4E45 542E
0x0000F120	646C 6C00 0000 4C6F	6164 4C69	6272 6172
0x0000F130	7941 0000 4765 7450	726F 6341	6464 7265
0x0000F140	7373 0000 5669 7274	7561 6C50	726F 7465
0x0000F150	6374 0000 5669 7274	7561 6C41	6C6C 6F63
0x0000F160	0000 5669 7274 7561	6C46 7265	6500 0000
0x0000F170	4578 6974 5072 6F63	4573 7300	0000 5265
0x0000F180	6743 6C6F 7365 4B65	7900 0000	5368 656C
0x0000F190	6C45 7865 6375 7465	4100 0000	5552 4C44
0x0000F1A0	6F77 6E6C 6F61 6454	6F46 696C	6541 0000
0x0000F1B0	5368 6F77 5769 6E64	6F77 0000	4465 6C65
0x0000F1C0	7465 5572 6C43 6163	6865 456E	7472 7900
0x0000F1D0	0000 0200 0C00 0000	8236 0000	0090 0200
0x0000F1E0	0C00 0000 D432 D832	0000 0000	0000 0000
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

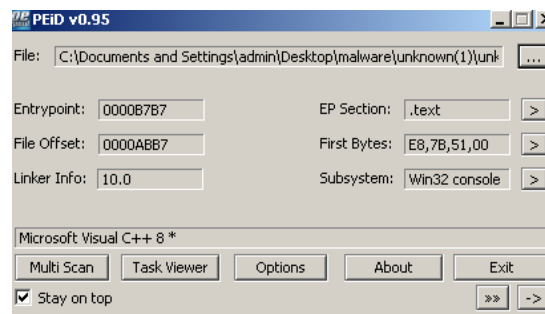


Figure.2.1.3.2.2 - PEiD Now shows Microsoft Visual C++ 8* and differing information.

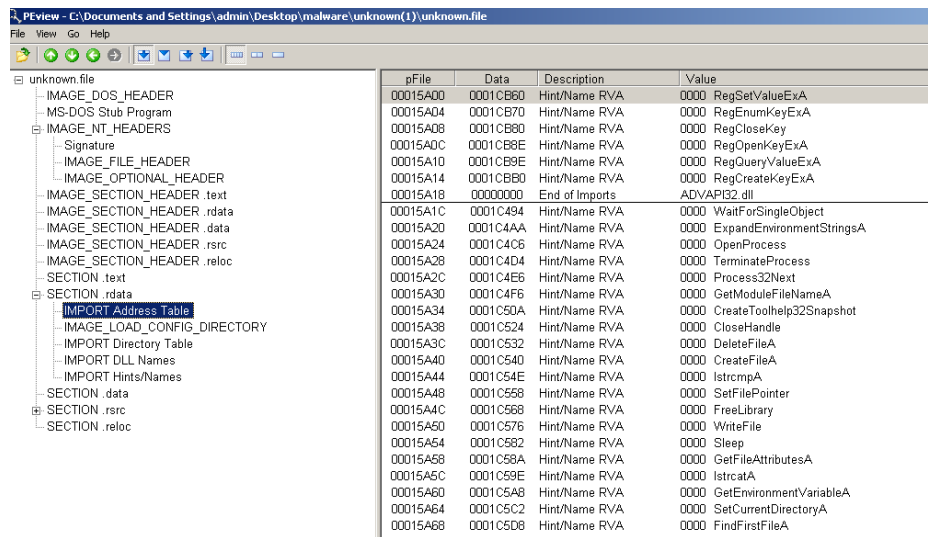


Figure.2.1.3.2.3 - PVIEW now has more information.

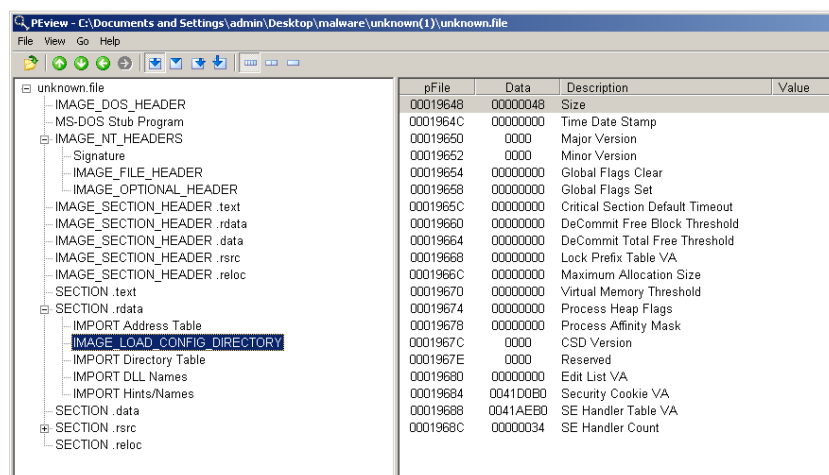


Figure.2.1.3.2.4 - PVIEW Config Directory.

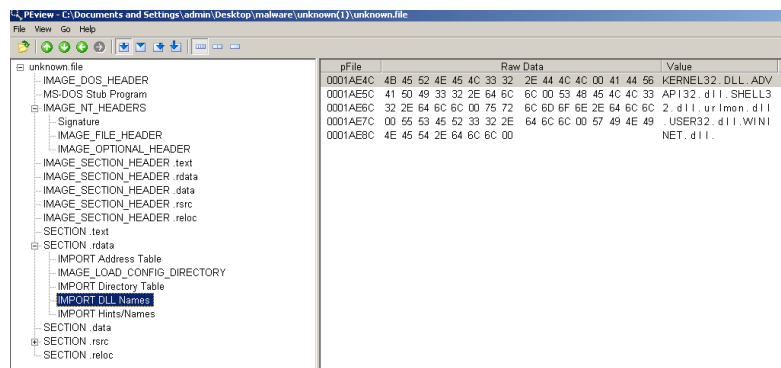


Figure.2.1.3.2.5 - PEview DLL Names.

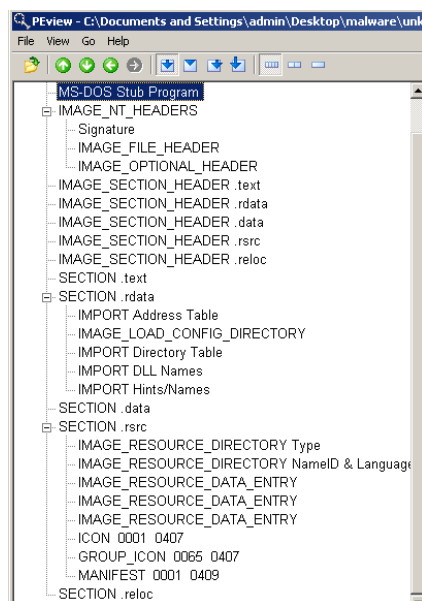


Figure.2.1.3.2.6 - The architecture is now different in PEview

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.

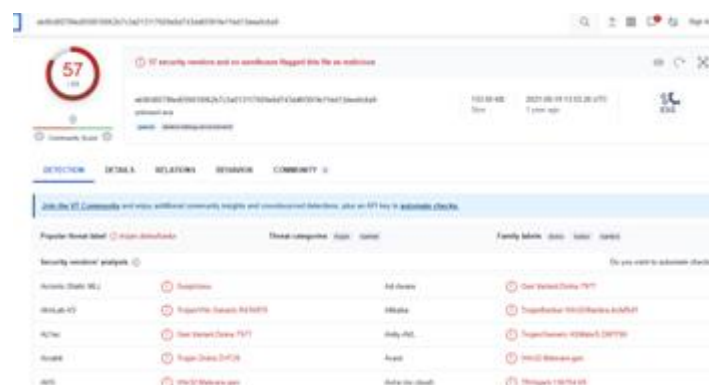


Figure.2.1.3.2.7 – VirusTotal output



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.

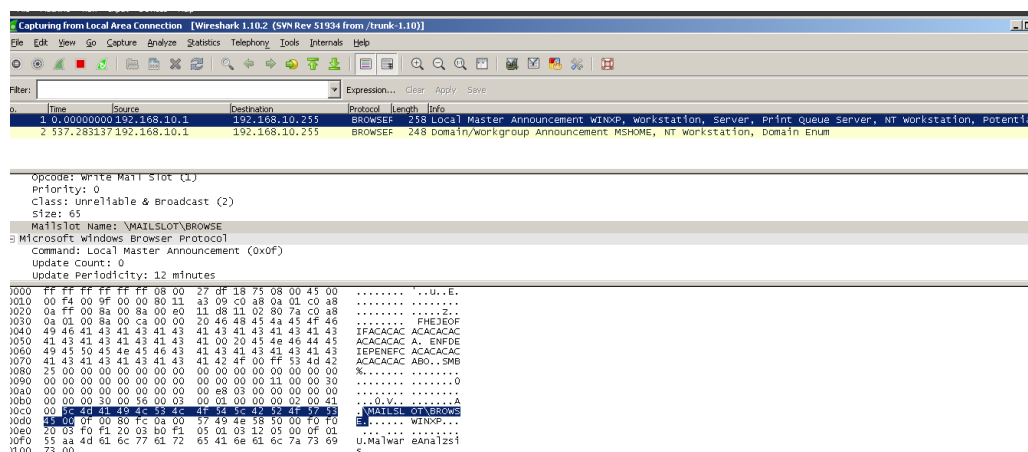


Figure.2.1.3.3.1 – Wireshark Capture for 'unknown' sample

```

FakeNet Version 1.0
Starting program, for help open a web browser and surf to any URL.
Press CTRL-C to exit.
Modifying local DNS Settings.
Scanning Installed Providers
Installing Layered Providers
Preparing To Reorder Installed Chains
Reordering Installed Chains
Moving New Protocol Order
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 8080.
Listening for traffic on port 1337.
Listening for SSL traffic on port 31337.
Listening for traffic on port 25.
Listening for SSL traffic on port 465.
Listening for ICMP traffic.
Listening for DNS traffic on port: 53.

NS Query Received.
Domain name: 255.10.168.192.in-addr.arpa
NS Response sent.

```

Figure.2.1.3.3.2 – FakeNet Capture for ‘unknown’ sample

Time...	Process Name	PID	Operation	Path	Result	Detail
8:48:4...	unknown.exe	3416	CloseFile	C:\WINDOWS\system32\wininet.dll	SUCCESS	
8:48:4...	unknown.exe	3416	RegOpenKey	HKLM\Software\Microsoft\Windows\Co...	NAME NOT FOUND	Desired Access: E...
8:48:4...	unknown.exe	3416	QueryOpen	C:\Documents and Settings\admin\Des...	NAME NOT FOUND	
8:48:4...	unknown.exe	3416	QueryOpen	C:\WINDOWS\WinSxS\x86_Microsoft...	SUCCESS	CreationTime: 10/3...
8:48:4...	unknown.exe	3416	CreateFile	C:\WINDOWS\WinSxS\x86_Microsoft...	SUCCESS	Desired Access: E...
8:48:4...	unknown.exe	3416	RegCreateKey	HKCU\SOFTWARE\Microsoft\Window...	SUCCESS	Desired Access: R...
8:48:4...	unknown.exe	3416	QueryOpen	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	CreationTime: 4/14...
8:48:4...	unknown.exe	3416	CreateFile	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	Desired Access: E...
8:48:4...	unknown.exe	3416	CreateFileMap...	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	SyncType: SyncTy...
8:48:4...	unknown.exe	3416	QueryStandard...	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	AllocationSize: 299...
8:48:4...	unknown.exe	3416	CreateFileMap...	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	SyncType: SyncTy...
8:48:4...	unknown.exe	3416	CloseFile	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	
8:48:4...	unknown.exe	3416	QueryOpen	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	CreationTime: 4/14...
8:48:4...	unknown.exe	3416	CreateFile	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	Desired Access: E...
8:48:4...	unknown.exe	3416	CreateFileMap...	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	SyncType: SyncTy...
8:48:4...	unknown.exe	3416	CreateFileMap...	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	SyncType: SyncTy...
8:48:4...	unknown.exe	3416	CloseFile	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	
8:48:4...	unknown.exe	3416	Load Image	C:\WINDOWS\system32\MSCTF.dll	SUCCESS	Image Base: 0x747...
8:48:4...	unknown.exe	3416	RegOpenKey	HKLM\Software\Microsoft\Windows N...	NAME NOT FOUND	Desired Access: R...
8:48:4...	unknown.exe	3416	QueryOpen	C:\WINDOWS\system32\ntdll.dll	SUCCESS	CreationTime: 4/14...
8:48:4...	unknown.exe	3416	QueryOpen	C:\WINDOWS\system32\ntlm32.dll	SUCCESS	CreationTime: 4/14...
8:48:4...	unknown.exe	3416	CreateFile	C:\WINDOWS\system32\ntlm32.dll	SUCCESS	Desired Access: E...
8:48:4...	unknown.exe	3416	CreateFileMap...	C:\WINDOWS\system32\ntlm32.dll	SUCCESS	SyncType: SyncTy...
8:48:4...	unknown.exe	3416	QueryStandard...	C:\WINDOWS\system32\ntlm32.dll	SUCCESS	All Desired Access: Execute/Traverse, Synchronize...
8:48:4...	unknown.exe	3416	CreateFileMap...	C:\WINDOWS\system32\ntlm32.dll	SUCCESS	Disposition: Open
8:48:4...	unknown.exe	3416	CloseFile	C:\WINDOWS\system32\ntlm32.dll	SUCCESS	Options: Synchronous IO Non-Alert, Non-Directory File
8:48:4...	unknown.exe	3416	RegOpenKey	HKLM\SOFTWARE\Microsoft\CTF\Co...	NAME NOT FOUND	Ds Attributes: n/a
8:48:4...	unknown.exe	3416	RegOpenKey	HKLM\SOFTWARE\Microsoft\CTF\Sys...	SUCCESS	Ds ShareMode: Read, Delete
8:48:4...	unknown.exe	3416	RegQueryValue	HKLM\SOFTWARE\Microsoft\CTF\Sys...	SUCCESS	AllocationSize: n/a
8:48:4...	unknown.exe	3416	RegCloseKey	HKLM\SOFTWARE\Microsoft\CTF\Sys...	SUCCESS	Type: REG_SZ, Le...
8:48:4...	unknown.exe	3416	RegOpenKey	HKCU\Keyboard Layout\Toggle	SUCCESS	Desired Access: R...
8:48:4...	unknown.exe	3416	RegQueryValue	HKCU\Keyboard Layout\Toggle\Langu...	SUCCESS	Type: REG_SZ, Le...

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.

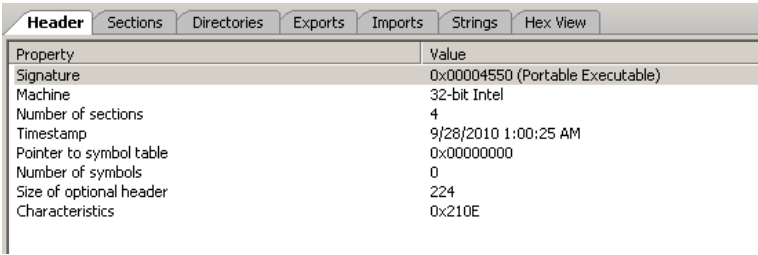
The exports and imports of this DLL sample suggest that it may be related to a Windows service that involves network and/or internet functionality. However, further analysis is necessary to fully understand the behavior and purpose of the sample. When investigating the Strings Tab, there are many function 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	ts ts					
117	1234567890123456					
118	getfile					
119	cmd.exe /c					
120	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/					
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 INet+, 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(ts) error %d					
137	Intranet Network Awareness (INet+)					
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(ts) KEY_QUERY_VALUE success.					
144	RegOpenKeyEx(ts) KEY_QUERY_VALUE error .					
145	SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost					
146	IPRIP					

Figure.3.3.2 – MiTec EXE Strings Tab Information



Header	Sections	Directories	Exports	Imports	Strings	Hex View
Property	Value					
Signature	0x00004550 (Portable Executable)					
Machine	32-bit Intel					
Number of sections	4					
Timestamp	9/28/2010 1:00:25 AM					
Pointer to symbol table	0x00000000					
Number of symbols	0					
Size of optional header	224					
Characteristics	0x210E					

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.

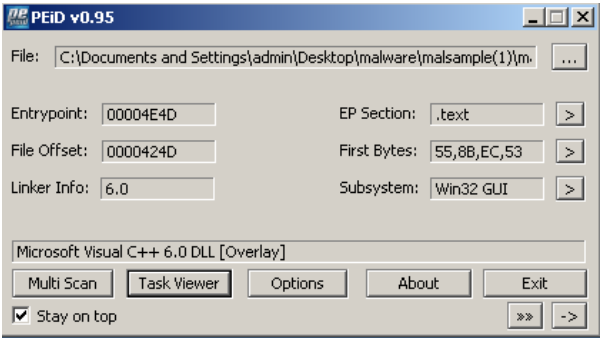
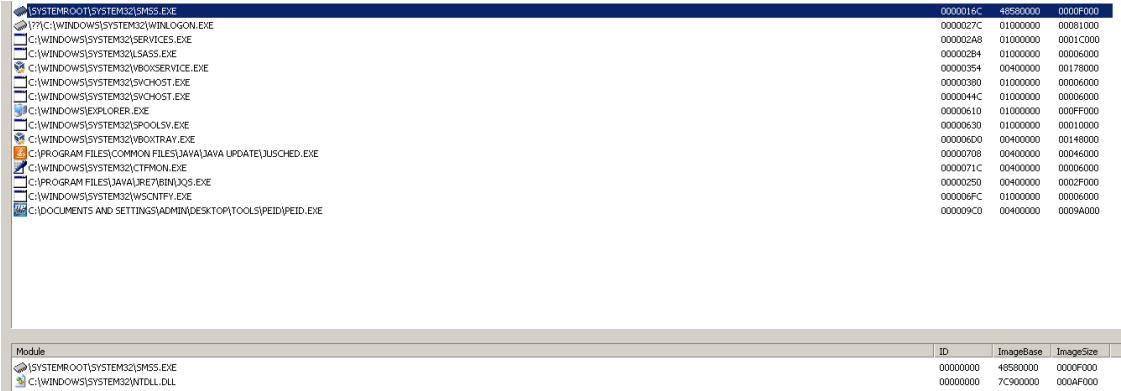


Figure.3.3.3 – PEiD Information



Module	ID	ImageBase	ImageSize
C:\SYSTEMROOT\SYSTEM32\SMSS.EXE	00000000	40500000	0000F000
C:\WINDOWS\SYSTEM32\NTDLL.DLL	00000000	7C900000	000AF000

Figure.3.3.4 - PEiD System Tasks Information

FileAlyzer

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

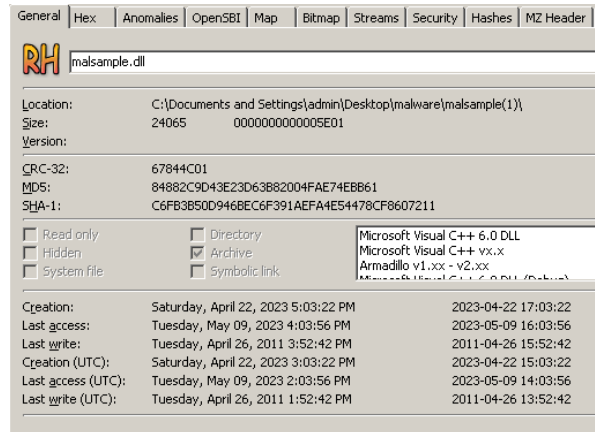


Figure.3.3.5 - FileAlyzer General Tab Information

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

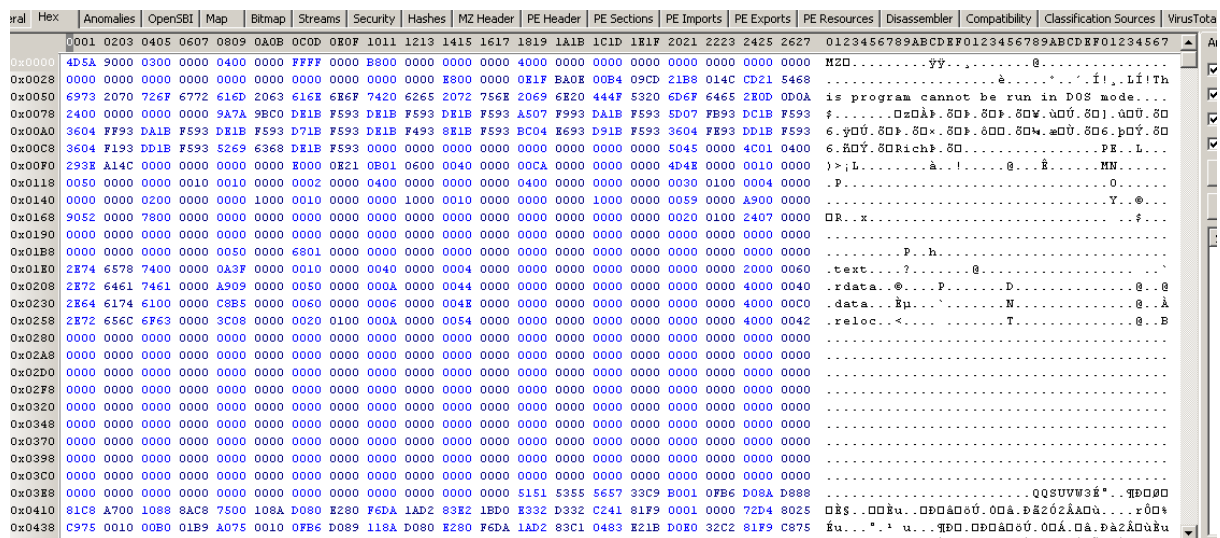


Figure.3.3.6 - FileAlyzer Hex Tab

The OpenSBI tab notes the file size could be incorrect.

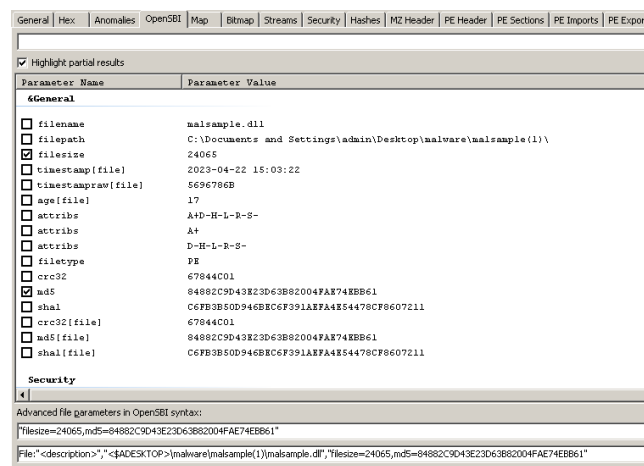


Figure.3.3.7 - FileAlyzer OpenSBI

General Hex Anomalies OpenSBI Map Bitmap Streams Security Hashes MZ Header PE Header PE Section	
<input checked="" type="checkbox"/> Highlight partial results	
Type	Value
admin (WINXP)	
ACE Type	ACCESS_ALLOWED_ACE_TYPE
Rights	FILE_ALL_ACCESS
SYSTEM (NT AUTHORITY)	
ACE Type	ACCESS_ALLOWED_ACE_TYPE
Rights	FILE_ALL_ACCESS
Administrators (BUILTIN)	
ACE Type	ACCESS_ALLOWED_ACE_TYPE
Rights	FILE_ALL_ACCESS

Figure.3.3.8 - FileAlyzer Security

Here the Security tabs shows an executable in the characteristics.

General	Hex	Anomalies	OpenSBI	Map	Bitmap	Streams	Security	Hashes	MZ Header	PE Header	PE Sections	PE Imports	PE Exports	PE
<input checked="" type="checkbox"/> Highlight partial results														
Header		Graph												
Field		Data			Details									
PE header														
Signature		00004550												
Machine		014C			Intel 386									
Number of sections		0004												
Time/Date stamp (local)		4CA13E29			2010-09-28 03:00:25									
Time/Date stamp (UTC)		4CA13E29			2010-09-28 01:00:25									
Pointer to symbol table		00000000												
Number of symbols		00000000												
Size of optional header		00E0												
Characteristics		210E			Executable, Line Numbers Stripped, Local Symbols Stripped, 32bit Machine Expected, DLL									

Figure.3.3.9 - FileAlyzer PE Header Tab

Section	VirtSize	VirtAddr	PhysSize	PhysAddr	Flags	CRC32	MD5	Characteristics
PE sections								
.text	00003E0A	00001000	00004000	00000400	60000020	DCR46908	188FA2537376E3FDE990F1A6EFA2DR14	* Code, Execute Access,...
.rdata	000009A9	00005000	00000A00	00004400	40000040	AFCl2E33	F8F5FC8EFF8AB73B75FE51A9FDFA8FED	Initialized Data, Read ...
.data	0000B5C8	00006000	00000600	00004E00	C0000040	269F2BF3	3826470C54FF98CA8BC080250599BD93	Initialized Data, Read ...
.reloc	0000083C	00012000	00000A00	00005400	42000040	CA8FE8BF	BE394AC1EBB22951E9E07CE1416A1524	Initialized Data, Disca...

Figure.3.3.10(a) - FileAlyzer PE Sections Tab

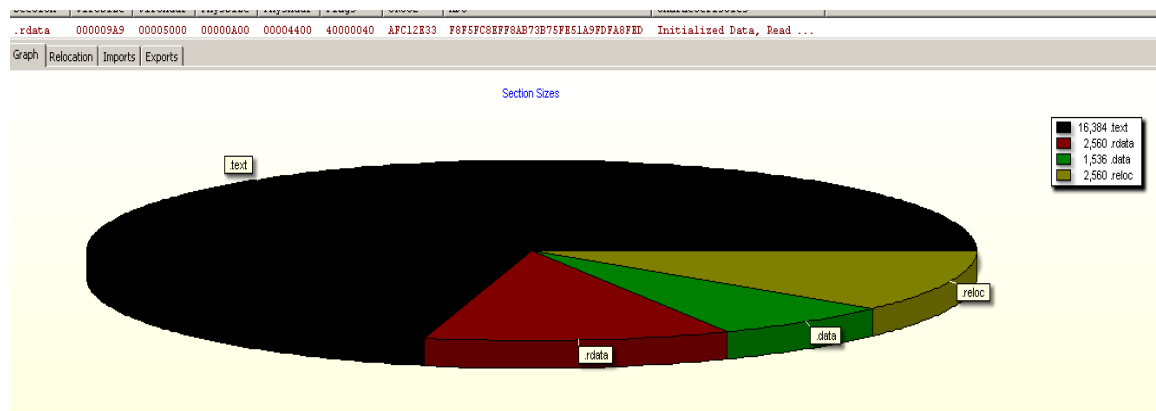


Figure.3.3.10(b) - FileAlyzer PE Sections Graph Tab

Graph	Relocation	Imports	Exports
Function	Address		
ADVAPI32.dll (12)			
OpenServiceA	0000568C		
DeleteService	0000567C		
RegOpenKeyExA	0000566C		
RegQueryValueExA	00005658		
RegCloseKey	0000564A		
OpenSCManagerA	00005638		
CreateServiceA	00005626		
CloseServiceHandle	00005610		
RegCreateKeyA	00005600		
RegSetValueExA	000055EE		
RegisterServiceCtrlHandlerA	000055D0		
SetServiceStatus	0000569C		

Graph	Relocation	Imports	Exports
Function	Address		
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		

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

Graph	Relocation	Imports	Exports
Function	Ordinal	Address	Physical
Header			
Characteristics	00000000		
Timestamp	4CA13E29	2010-09-28	01:00:25
Version	0.0		
Name RVA	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.3.3.10(d) - FileAlyzer PE Sections Exports Tab

General	Hex	Anomalies	OpenSBI	Map	Bitmap	Streams	Security	Hashes	MZ Header	PE Header	PE Sections	PE Imports	PE Exports	PI
<input checked="" type="checkbox"/> Highlight partial results														
Imports Graph														
Function	Address	95-2 B	NT4 SP1	NT4 SP3	NT4 SP4	NT4 SP6	98 A	ME						
ADVAPI32.dll (12)														
OpenServiceA	0000568C	+	+	+	+	+	+	+						
DeleteService	0000567C	+	+	+	+	+	+	+						
RegOpenKeyExA	0000566C	+	+	+	+	+	+	+						
RegQueryValueExA	00005658	+	+	+	+	+	+	+						
RegCloseKey	0000564A	+	+	+	+	+	+	+						
OpenSCManagerA	00005638	+	+	+	+	+	+	+						
CreateServiceA	00005626	+	+	+	+	+	+	+						
CloseServiceHandle	00005610	+	+	+	+	+	+	+						
RegCreateKeyA	00005600	+	+	+	+	+	+	+						
RegSetValueExA	000055EE	+	+	+	+	+	+	+						
RegisterServiceCtrlHandlerA	000055D0	+	+	+	+	+	+	+						
SetServiceStatus	0000569C	+	+	+	+	+	+	+						

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
<input checked="" type="checkbox"/> 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	00003806						
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:	0x00004295														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	RDT 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

Service	Status	Timestamp
Blacklists		
Malware Hash Registry (Team Cymru)	n/a	n/a
Comodo	unlisted	2023-05-09 16:33:25
Mixed lists		
Google	unlisted	2023-05-09 16:33:25
Bing	unlisted	2023-05-09 16:33:25
Yahoo	unlisted	2023-05-09 16:33:25

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.

```

C:\Documents and Settings\admin\Desktop\Tools\upx\upx391w>upx -d malsample.dll
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
-----
24065 <-      12289      51.07%      win32/pe      malsample.dll

Unpacked 1 file.
  
```

Figure.3.4.1 – Unpacked malsample

RegShot

Here I have ran the service ‘install’ and documented the findings. Command: `run32dll.exe malsample.dll,install`

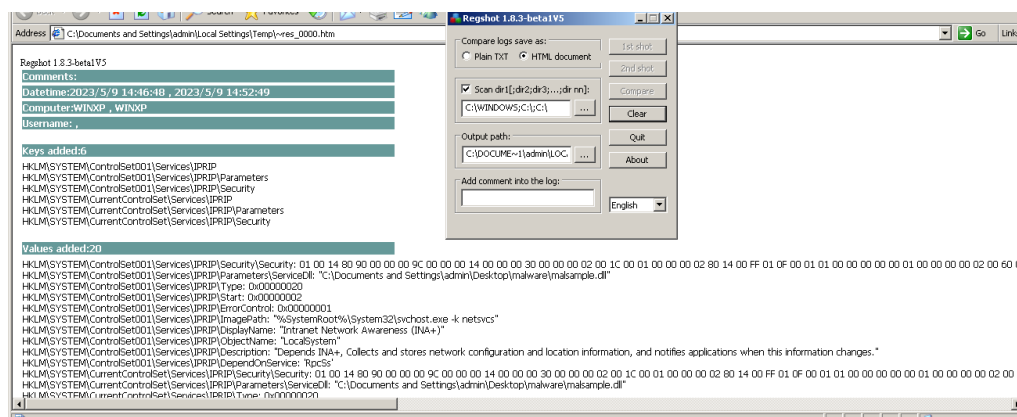


Figure.3.4.2 - RegShot Capture of ‘malsample.dll’ 1

```

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    <DIR>          224,078 cw_pdf_files<1>.7z
05/09/2023  04:19 PM    <DIR>          malsample<1>
04/19/2023  07:27 PM    <DIR>          10,418 malsample<1>.7z
04/22/2023  08:32 PM    <DIR>          unknown<1>
04/19/2023  07:26 PM    <DIR>          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, Install
all

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

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

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

```

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

FakeNet

```

C:\Documents and Settings\admin\Desktop\Tools\FakeNet\Tools\FakeNet.exe
[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 8080.]
[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

```

C:\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    <DIR>          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    <DIR>          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

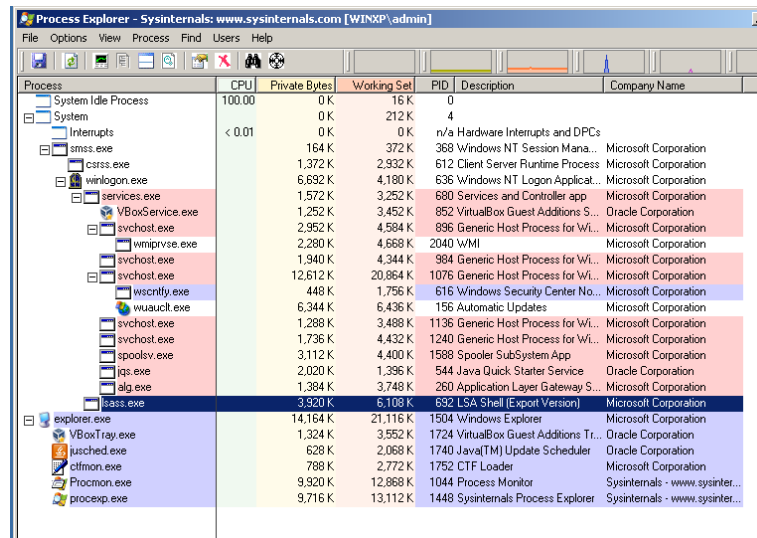
C:\Documents and Settings\admin\Desktop\malsample\malsample>

```

Figure.3.4.6 – dll run command

Process Explorer

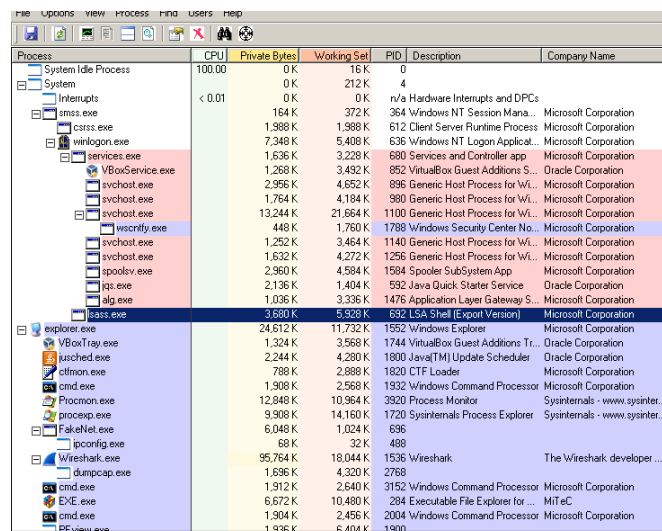
No additional services seemed to have started, peep the before and after below. I did note that when investigated further within the Internet Explorer running process, the same strings were present in the strings tab of this process within Process Explorer.



Process Explorer - Sysinternals: www.sysinternals.com [WINXP\admin]

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
System Idle Process	100.00	0 K	16 K	0		
System		0 K	212 K	4		
Interrupts	< 0.01	0 K	0 K	n/a	Hardware Interrupts and DPCs	
csrss.exe		164 K	372 K	368	Windows NT Session Mana...	Microsoft Corporation
winlogon.exe		1,372 K	2,932 K	612	Client Server Runtime Process	Microsoft Corporation
services.exe		6,692 K	4,180 K	636	Windows NT Logon Applicat...	Microsoft Corporation
VBoxService.exe		1,572 K	3,252 K	680	Services and Controller app	Microsoft Corporation
svchost.exe		1,252 K	3,452 K	852	VirtualBox Guest Additions S...	Oracle Corporation
svchost.exe		2,952 K	4,584 K	896	Generic Host Process for Wi...	Microsoft Corporation
wmiiprvse.exe		2,280 K	4,668 K	2040	WMI	Microsoft Corporation
svchost.exe		1,940 K	4,344 K	984	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe		12,612 K	20,864 K	1076	Generic Host Process for Wi...	Microsoft Corporation
wsnscntfy.exe		448 K	1,756 K	616	Windows Security Center No...	Microsoft Corporation
wuauclt.exe		6,344 K	6,436 K	156	Automatic Updates	Microsoft Corporation
svchost.exe		1,288 K	3,488 K	1136	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe		1,736 K	4,432 K	1240	Generic Host Process for Wi...	Microsoft Corporation
spoolsv.exe		3,112 K	4,400 K	1588	Spooler SubSystem App	Microsoft Corporation
iqs.exe		2,020 K	1,396 K	544	Java Quick Starter Service	Oracle Corporation
alg.exe		1,384 K	3,748 K	260	Application Layer Gateway S...	Microsoft Corporation
lsass.exe		3,920 K	6,108 K	692	LSA Shell (Export Version)	Microsoft Corporation
explorer.exe		14,164 K	21,116 K	1504	Windows Explorer	Microsoft Corporation
VBoxTray.exe		1,324 K	3,552 K	1724	VirtualBox Guest Additions Tr...	Oracle Corporation
tsched.exe		628 K	2,068 K	1740	Java(TM) Update Scheduler	Oracle Corporation
ctfmon.exe		788 K	2,772 K	1752	CTF Loader	Microsoft Corporation
Procmon.exe		9,920 K	12,868 K	1044	Process Monitor	Sysinternals - www.sysinter...
procexp.exe		9,716 K	13,112 K	1448	Sysinternals Process Explorer	Sysinternals - www.sysinter...

Figure.3.4.7 – Process Explore Prior to Execution



Process Explorer - Sysinternals: www.sysinternals.com [WINXP\admin]

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
System Idle Process	100.00	0 K	16 K	0		
System		0 K	212 K	4		
Interrupts	< 0.01	0 K	0 K	n/a	Hardware Interrupts and DPCs	
csrss.exe		1,988 K	1,988 K	364	Windows NT Session Mana...	Microsoft Corporation
winlogon.exe		7,348 K	5,408 K	612	Client Server Runtime Process	Microsoft Corporation
services.exe		1,536 K	3,228 K	636	Windows NT Logon Applicat...	Microsoft Corporation
VBoxService.exe		1,268 K	3,492 K	680	Services and Controller app	Microsoft Corporation
svchost.exe		2,956 K	4,652 K	852	VirtualBox Guest Additions S...	Oracle Corporation
svchost.exe		1,764 K	4,184 K	896	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe		13,244 K	21,664 K	980	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe		448 K	1,760 K	1100	Generic Host Process for Wi...	Microsoft Corporation
wsnscntfy.exe		1,252 K	3,464 K	1788	Windows Security Center No...	Microsoft Corporation
svchost.exe		1,632 K	4,272 K	1140	Generic Host Process for Wi...	Microsoft Corporation
svchost.exe		1,632 K	4,272 K	1256	Generic Host Process for Wi...	Microsoft Corporation
spoolsv.exe		2,960 K	4,584 K	1584	Spooler SubSystem App	Microsoft Corporation
iqs.exe		2,136 K	1,404 K	592	Java Quick Starter Service	Oracle Corporation
alg.exe		1,036 K	3,336 K	1476	Application Layer Gateway S...	Microsoft Corporation
lsass.exe		3,920 K	5,920 K	692	LSA Shell (Export Version)	Microsoft Corporation
explorer.exe		24,612 K	11,732 K	1552	Windows Explorer	Microsoft Corporation
VBoxTray.exe		1,324 K	3,568 K	1744	VirtualBox Guest Additions Tr...	Oracle Corporation
tsched.exe		2,244 K	4,280 K	1800	Java(TM) Update Scheduler	Oracle Corporation
ctfmon.exe		788 K	2,888 K	1820	CTF Loader	Microsoft Corporation
cmd.exe		1,908 K	2,568 K	1932	Windows Command Processor	Microsoft Corporation
Procmon.exe		12,848 K	10,964 K	3920	Process Monitor	Sysinternals - www.sysinter...
procexp.exe		9,908 K	14,160 K	1720	Sysinternals Process Explorer	Sysinternals - www.sysinter...
FakeNet.exe		6,048 K	1,024 K	696		
ipconfig.exe		68 K	32 K	488		
Wireshark.exe		95,764 K	18,044 K	1536	Wireshark	The Wireshark developer ...
dumpcap.exe		1,596 K	4,320 K	2768		
cmd.exe		1,912 K	2,640 K	3152	Windows Command Processor	Microsoft Corporation
ExE.exe		6,672 K	10,480 K	284	Executable File Explorer for ...	MITeC
cmd.exe		1,904 K	2,456 K	2004	Windows Command Processor	Microsoft Corporation
PC utilities.exe		1,936 K	6,404 K	1900		

Figure.3.4.8 – Process Explore After to Execution

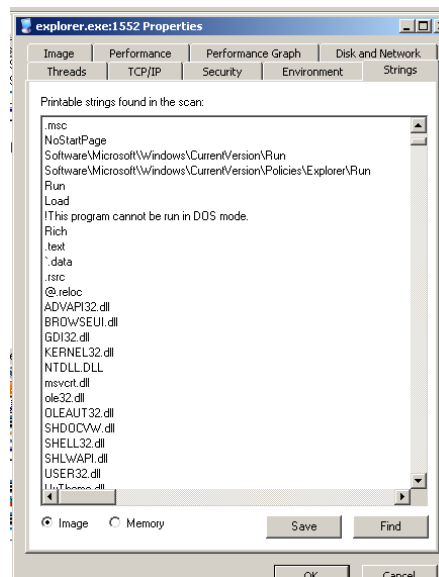
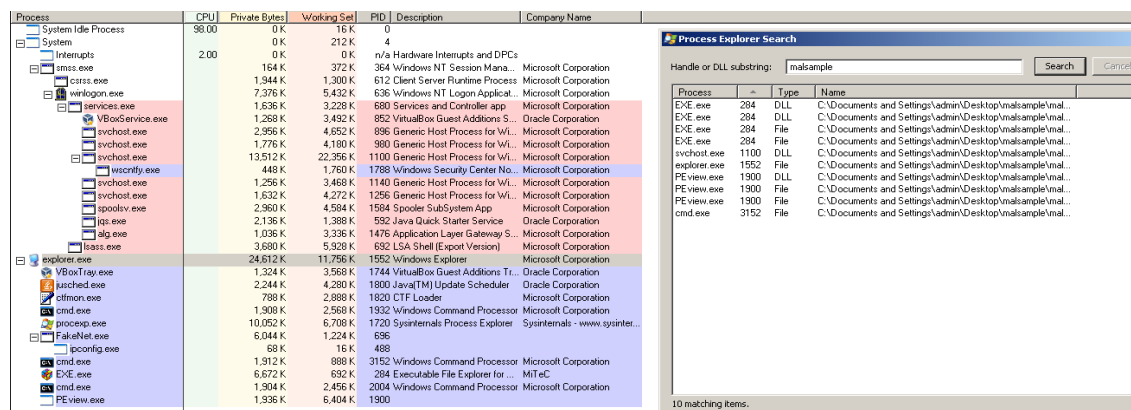


Figure.3.4.9 – Process Explorer Internet Explorer Properties, then Strings tab

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.

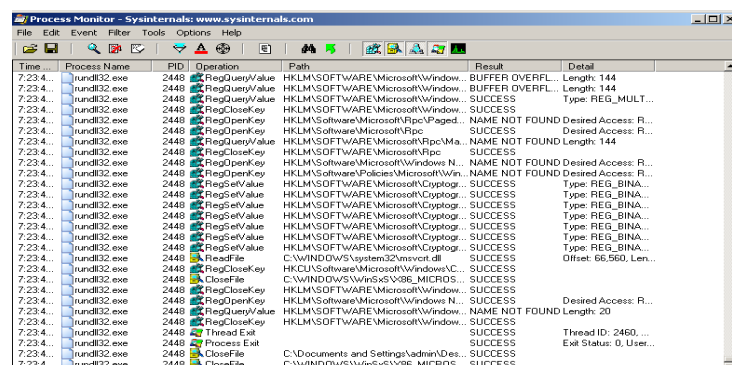


Figure.3.4.10 – Process Monitor Filtered to rundll32.exe

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 Analysis’ discovered in the analysis stage.

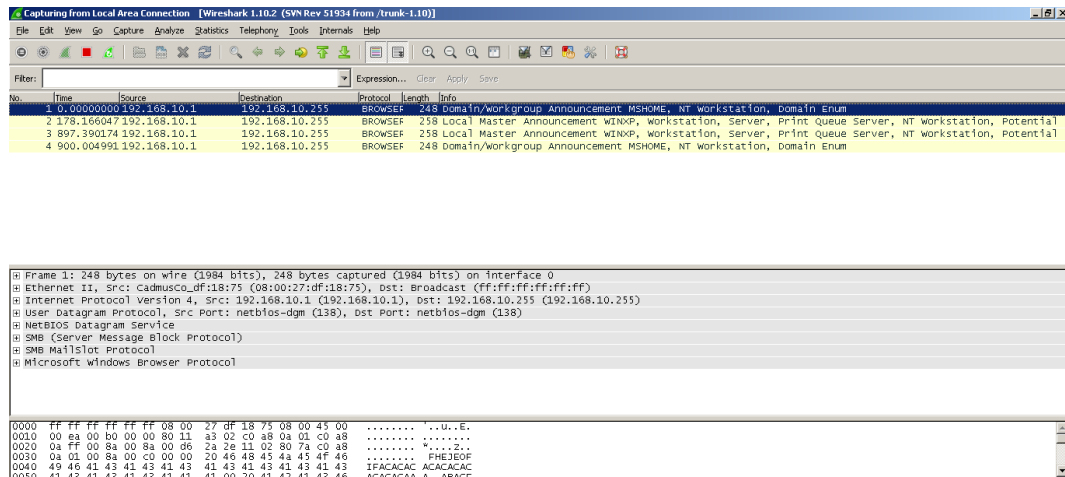


Figure.3.5.1 – Wireshark after installing service (1)

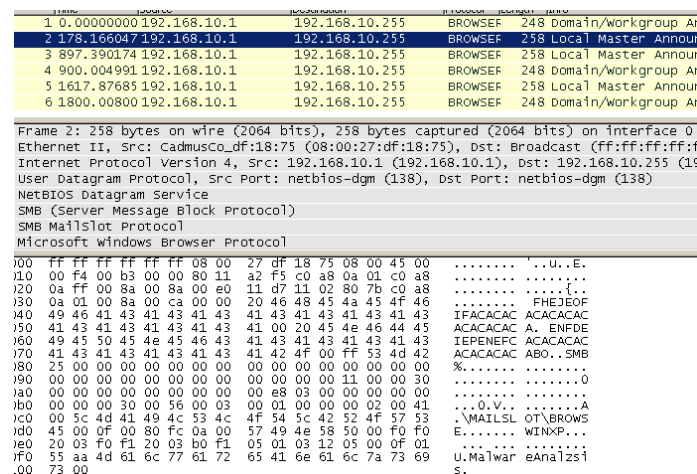


Figure.3.5.2 – Wireshark after installing service (2)

3.6 – Reverse Engineering of ‘malsample.dll’

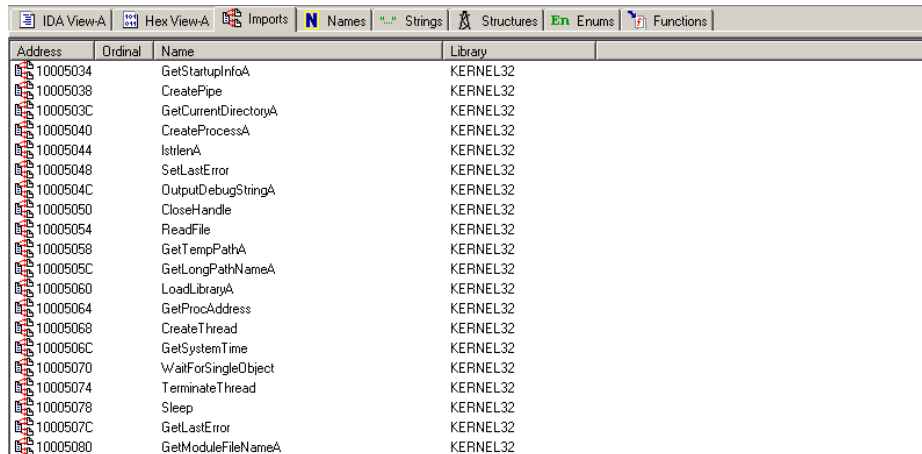
(a) How many functions are exported by the DLL? There are six exported functions; Install, ServiceMain, UninstallService, InstallA, and DllEntryPoint.

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

Name	Address	Ordinal
Install	10004706	1
ServiceMain	10003196	2
UninstallService	10004B18	3
installA	10004B0B	4
uninstallA	10004C2B	5
DllEntryPoint	10004E4D	

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

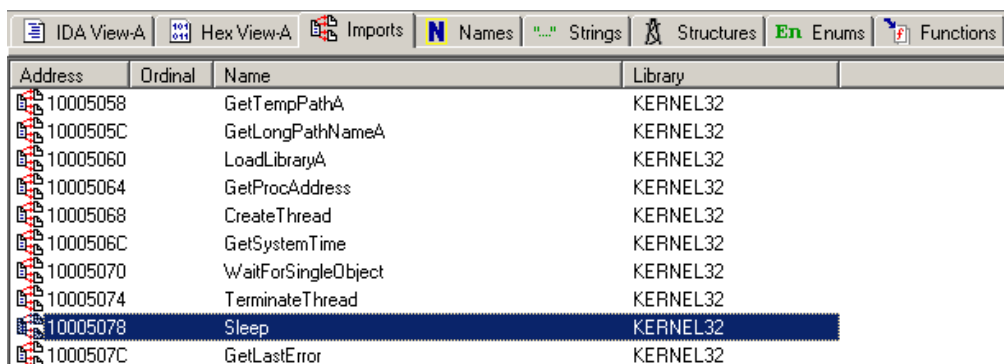
(c) How many functions call the kernel32 API LoadLibrary? I documented twenty occurrences that kernel32.dll LoadLibrary is called.



Address	Ordinal	Name	Library
10005034		GetStartupInfoA	KERNEL32
10005038		CreatePipe	KERNEL32
1000503C		GetCurrentDirectoryA	KERNEL32
10005040		CreateProcessA	KERNEL32
10005044		IsUserA	KERNEL32
10005048		SetLastError	KERNEL32
1000504C		OutputDebugStringA	KERNEL32
10005050		CloseHandle	KERNEL32
10005054		ReadFile	KERNEL32
10005058		GetTempPathA	KERNEL32
1000505C		GetLongPathNameA	KERNEL32
10005060		LoadLibraryA	KERNEL32
10005064		GetProcAddress	KERNEL32
10005068		CreateThread	KERNEL32
1000506C		GetSystemTime	KERNEL32
10005070		WaitForSingleObject	KERNEL32
10005074		TerminateThread	KERNEL32
10005078		Sleep	KERNEL32
1000507C		GetLastError	KERNEL32
10005080		GetModuleFileNameA	KERNEL32

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.

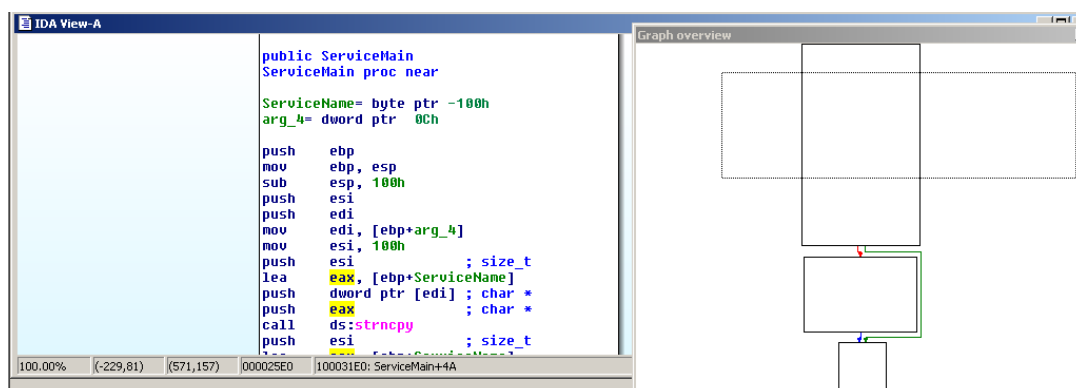


Address	Ordinal	Name	Library
10005058		GetTempPathA	KERNEL32
1000505C		GetLongPathNameA	KERNEL32
10005060		LoadLibraryA	KERNEL32
10005064		GetProcAddress	KERNEL32
10005068		CreateThread	KERNEL32
1000506C		GetSystemTime	KERNEL32
10005070		WaitForSingleObject	KERNEL32
10005074		TerminateThread	KERNEL32
10005078		Sleep	KERNEL32
1000507C		GetLastError	KERNEL32

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

5. Navigate to the ServiceMain function.

(a) Show the graph view of the function



IDA View-A

```

public ServiceMain
ServiceMain proc near
ServiceName= byte ptr -100h
arg_4= dword ptr 0Ch

push    ebp
mov     ebp, esp
sub     esp, 100h
push    esi
push    edi
mov     edi, [ebp+arg_4]
mov     esi, 100h
push    esi ; size_t
lea     eax, [ebp+ServiceName]
push    dword ptr [edi] ; char *
push    eax ; char *
call    ds:strcmp
push    esi ; size_t

```

Graph overview

The graph overview shows a single block representing the function body, with a red arrow indicating the entry point and a blue arrow indicating the exit point.

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?

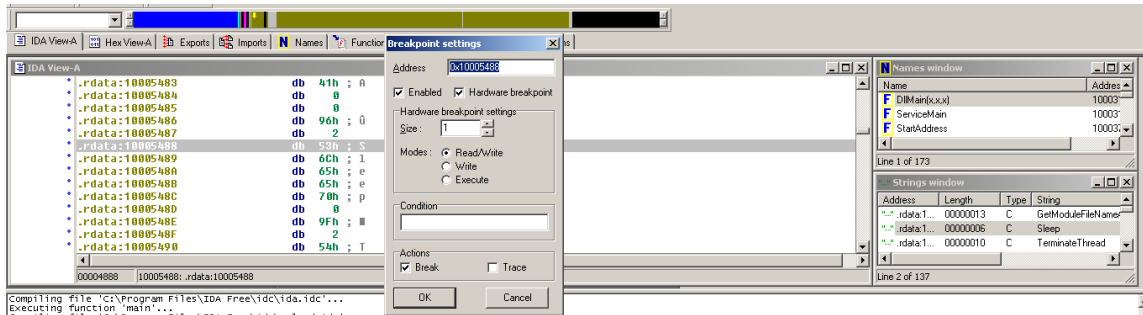


Figure.3.6.4(a) – Answer to question 5b

Exports		
Name	Address	Ordinal
Install	10004706	1
ServiceMain	10003196	2
UninstallService	10004818	3
installA	10004808	4
uninstallA	10004C2B	5
DllEntryPoint	10004E4D	

Figure.3.6.5 – Exports (IDA)

Function name	Segment	Start	Length	R	F	L	S	B	T	=
sub_100015A4	text	100015A4	00000070	R	.	.	.	B	.	.
sub_10002314	text	10002314	00000039	R	.	.	.	B	.	.
DllMain(xxx)	text	1000314D	00000049	R	T	.
ServiceMain	text	10003196	00000084	R	.	.	.	B	.	.
sub_1000321A	text	1000321A	0000006C	R	.	.	.	B	.	.
sub_10003286	text	10003286	0000018F	R
sub_10003415	text	10003415	000002FA	R	.	.	.	B	T	.
sub_1000370F	text	1000370F	00000088	R	.	.	.	B	.	.
StartAddress	text	10003797	0000027C	R	.	.	.	B	T	.
sub_10003A13	text	10003A13	0000048D	R	.	.	.	B	T	.
sub_10003ED0	text	10003ED0	00000253	R	.	.	.	B	T	.
sub_10004123	text	10004123	000001F4	R	.	.	.	B	T	.
sub_10004317	text	10004317	0000004C	R
sub_10004363	text	10004363	000001E8	R	.	.	.	B	T	.
sub_1000454E	text	1000454E	00000106	R
sub_10004654	text	10004654	00000075	R	.	.	.	B	T	.
sub_100046C9	text	100046C9	0000003D	R	T	.
Install	text	10004706	0000004D	R
installA	text	10004608	0000000D	R
UninstallService	text	10004618	00000113	R	.	.	.	B	T	.
uninstallA	text	10004C2B	0000000D	R
sub_10004C38	text	10004C38	00000049	R	.	.	.	B	.	.
HandlerProc	text	10004C80	00000077	R	T	.
_WSAFDIsSet	text	10004CF8	00000006	R	T	.
memset	text	10004CFE	00000006	R	T	.
strcpy	text	10004D04	00000006	R	T	.
strcpy	text	10004D0A	00000006	R	T	.
operator delete(void*)	text	10004D10	00000006	R	T	.
operator new(uint)	text	10004D16	00000006	R	T	.
strlen	text	10004D1C	00000006	R	T	.
_alloca_probe	text	10004D30	0000002F	R	.	L
sub_10004D5F	text	10004D5F	0000001C	R
_EH_prolog	text	10004D90	00000006	R
_CxxThrowException	text	10004D96	00000006	R
_except_handler3	text	10004D9C	00000006	R
_CRT_INIT(xxx)	text	10004DA2	000000A8	R	.	L
DllEntryPoint	text	10004E4D	0000009D	R	.	L	.	B	T	.
type_info::~type_info(void)	text	10004EEA	00000006	R
_initterm	text	10004EF0	00000006	R

Figure.3.6.6 – Functions (IDA)

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

Figure.3.6.7 – Strings (IDA)

IDA View-A	Hex View-A	Exports	Imports	Names	Functions	Strings	Structures
Address	Length	Type	String				
00000011	00000011	C	UninstallService				
00000009	00000009	C	installA				
0000000B	0000000B	C	uninstallA				
0000000D	0000000D	C	Y29ubmVjdA==				
0000001D	0000001D	C	practicalmalwareanalysis.com				
0000000B	0000000B	C	serve.html				
0000000D	0000000D	C	dW5zdXBwb3J0				
00000009	00000009	C	c2xZKA=				
00000005	00000005	C	Y21k				
00000009	00000009	C	cXVpdA==				
00000011	00000011	C	Windows XP 6.11				
0000000F	0000000F	C	CreateProcessA				
0000000D	0000000D	C	kernel32.dll				
00000005	00000005	C	.exe				
00000009	00000009	C	HTTP/1.1				
00000006	00000006	C	%s %s				
00000011	00000011	C	1234567890123456				
00000005	00000005	C	quit				
00000005	00000005	C	exit				
00000008	00000008	C	getFile				
0000000C	0000000C	C	cmd.exe /c				
00000041	00000041	C	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345...				
00000005	00000005	C	->				
00000005	00000005	C	<!--				
00000005	00000005	C	.PAX				
00000010	00000010	C	DependOnService				
00000006	00000006	C	RpcSs				
00000008	00000008	C	ServiceDll				
00000021	00000021	C	GetModuleFileName() get dll path				
00000008	00000008	C	Parameters				
00000005	00000005	C	Type				
00000006	00000006	C	Start				
00000008	00000008	C	ObjectName				
0000000C	0000000C	C	LocalSystem				
0000000D	0000000D	C	ErrorControl				
0000000C	0000000C	C	DisplayName				
0000000C	0000000C	C	Description				
00000008	00000008	C	Depends INA+, Collects and stores network configuration and location infor...				
0000000A	0000000A	C	ImagePath				
00000026	00000026	C	%SystemRoot%\System32\svchost.exe -k				

Figure.3.6.8 – Strings 2 (IDA)

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

Figure.3.6.9 – Strings 3 (IDA)

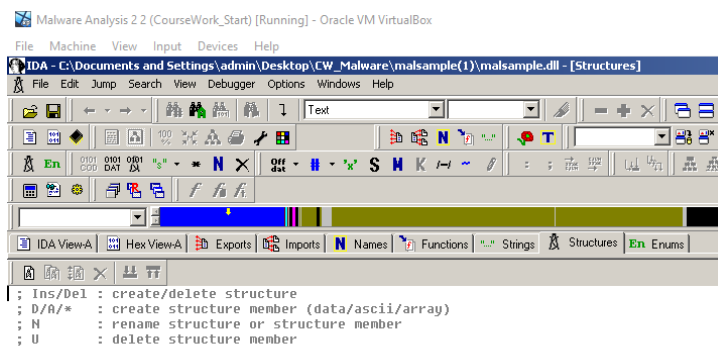


Figure.3.6.10 - Structures (IDA)

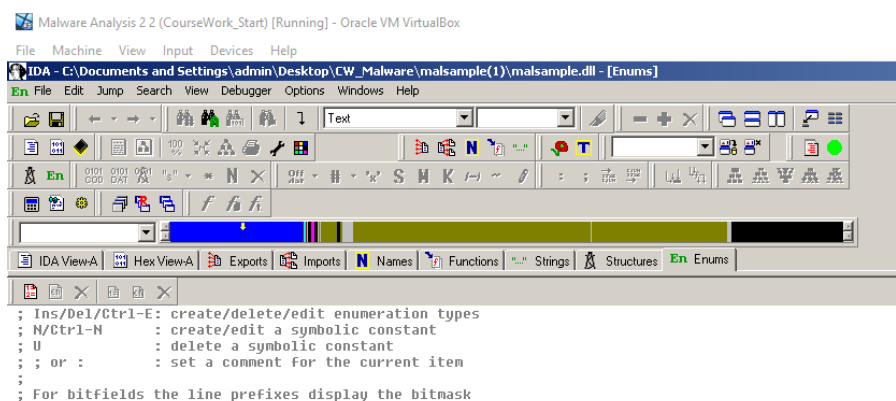


Figure. 3.6.11– Enums (IDA)

Graph	Relocation	Imports	Exports
Function	Address		
ADVAPI32.dll (12)			
OpenServiceA	0000568C		
DeleteService	0000567C		
RegOpenKeyExA	0000566C		
RegQueryValueExA	00005658		
RegCloseKey	0000564A		
OpenSCManagerA	00005638		
CreateServiceA	00005626		
CloseServiceHandle	00005610		
RegCreateKeyA	00005600		
RegSetValueExA	000055EE		
RegisterServiceCtrlHandlerA	000055D0		
SetServiceStatus	0000569C		

Graph	Relocation	Imports	Exports
Function	Address		
GetStartupInfoA	00005548		
CreatePipe	0000555A		
GetCurrentDirectoryA	00005568		
CreateProcessA	00005536		
IsntlnA	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		

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).