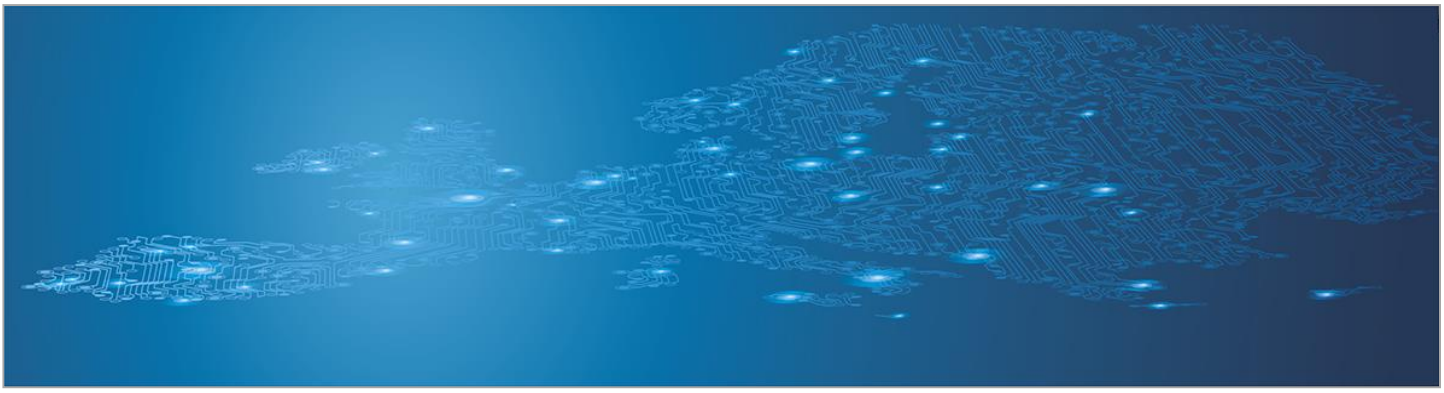


Artifact analysis fundamentals

Artifact analysis training material

November 2014

European Union Agency for Network and Information Security www.enisa.europa.eu

About ENISA

The European Union Agency for Network and Information Security (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

Authors

This document was created by Lauri Palkmets, Cosmin Ciobanu, Yonas Leguesse, and Christos Sidiropoulos in consultation with DFN-CERT Services[1] (Germany), ComCERT[2] (Poland), and S-CURE[3] (The Netherlands).

Contact

For contacting the authors please use cert-relations@enisa.europa.eu

For media enquires about this paper, please use press@enisa.europa.eu

Acknowledgements

ENISA wants to thank all institutions and persons who contributed to this document. A special 'Thank You' goes to Todor Dragostinov from ESMIS, Bulgaria.

[1] Klaus Möller, and Mirko Wollenberg
[2] Mirosław Maj, Tomasz Chlebowski, Krystian Kochanowski, Dawid Osojca, Paweł Weżgowiec, and Adam Ziaja
[3] Michael Potter, Alan Robinson, and Don Stikvoort

Table of Contents

Main Objective	Present the trainees malicious artifact analysis fundamentals and various types of analyses. Present how to safely execute suspicious code in the controlled environment along with most important security precautions. Teach the trainees how to perform basic static, behavioural, network and automatic analyses – what tools can be used, what to look for, what can be found. Give the trainees the opportunity to use various popular tools during the analyses and let them decide what tools are best suited for different type of analyses. Present common malicious software behaviours and patterns – which can be later used to create proper signature.	
Targeted Audience	The exercise is dedicated to CERT staff involved in analysis of malicious artifacts. The exercise should be also helpful to CERT staff involved in doing quick assessment of encountered new threats, especially those associated with suspicious executable files.	
Total Duration	8.0 hours	
Time Schedule	Introduction to the exercise and tools overview	1.0 hours
	Task 1: Basic static analysis	1.5 hours
	Task 2: Behavioural analysis	2.0 hours
	Task 3: Network analysis	1.5 hours
	Task 4: Automatic analysis	1.5 hours
	Summary of the exercise	0.5 hours
Frequency	It is advisable to organise this exercise when new team members who are involved in the analysis of malicious artifacts joins CERT.	

1 General description

The primary purpose of this exercise is to gather information about artifacts collected in previous exercises. At the beginning, participants will learn how to use basic static analysis techniques to perform a preliminary study of the sample. Using methods such as strings analysis, portable executable (PE) headers analysis, import address table (IAT) analysis or resources analysis, participants will try to determine some of the artifacts' functionality. At the same time the participants should look for any special features of the analysed samples which may be later used to create signatures.

In the second stage, participants will perform behavioural analysis in which they execute samples in a controlled environment. Then they will observe any changes taking place in the operating system: which processes are created, what changes are made to the file system or the system registry, and if there would be any indicators of rootkit activity. Next, using all gathered information, participants will try to answer how the analysed samples behave after being executed and what would be the indicators of an infected system.

In the next stage participants will learn how to perform basic network analysis using various tools and methods to capture network traffic. During this part of the exercise participants will try to detect traces of the malware activity in the network traffic. Based on the analysis results, they will try to deduce some of the artifact functionality and answer if there are any characteristic traffic patterns.

At the end, after learning basic static analysis, behavioural analysis and network analysis, participants will perform automatic analysis using the Cuckoo Sandbox appliance. In this way participants will get the opportunity to compare manual analysis techniques with the automatic analysis and learn what are the advantages and disadvantages of using both of them.

The exercise is performed using Microsoft Windows operating system. Analysed artifacts are in portable executable (PE) file format.

2 Introduction

2.1 Malware analysis fundamentals

Malware analysis is a process that uses various tools and techniques to determine how malicious code is working. Unfortunately there is no single algorithm to indicate how to analyse such code. Various approaches are usually needed including static analysis, behavioural analysis, executable debugging or analysis of disassembled code. Moreover each analyst usually tends to have his or her own favourite techniques and preferred tools.

Because malware analysis is usually a complex task, it is always important to have a clear goal in mind. Some common analysis goals are: determining infection indicators in order to detect other infected computers, determining malware propagation mechanism to more effectively prevent future infections or getting to know malicious code functionality to assess the risk associated with potential infection.

Malware samples are usually quite complex. Some samples are fairly easy to analyse while others require adeep knowledge of system internals and advanced reverse engineering skills to analyse. In general, to perform basic malware analysis some basic system administration knowledge and programming background is needed. And on the network level it is good to know the network stack and have some knowledge about popular network protocols (ICMP, TCP, UDP, HTTP, FTP, SMTP, IRC, etc.).

2.2 Various approaches to malware analysis

There are various approaches to malware analysis each having a different purpose and application. Usually more than one approach is used to gather necessary information about the sample in question. All approaches are described in the context of analysing Windows executable files in PE format, which this exercise uses.

Basic static analysis	In this analysis, the file structure of a malware sample is analysed without executing malicious code. The goal of this analysis is to gather information about potential malware functionality and any characteristic file features that could later be used to create malware signature. During the analysis various elements are checked such as strings list, import and export tables, list of file sections, file resources and PE headers. The file is also checked for signatures of well-known packers and searched for any embedded objects (images, executables, etc.).
Behavioural analysis	In this analysis, malicious code is intentionally executed in a controlled environment to observe what changes it makes to the operating system. We monitor elements like file system changes, changes in the Windows registry, changes on the process list, system resources usage, as well as any other visible anomalies (e.g. disappearing files). The operating system is also scanned for signs of rootkits activity. Based on observed system changes the analyst tries to deduce some of the malware's functionality. Behavioural analysis also allows us to determine the malware's persistence mechanism and

	indicators of infection. Knowledge about the persistence mechanism and infection indicators might be used to identify other infected workstations and to disinfect them.
Network analysis	Network analysis is usually performed alongside behavioural analysis. During network analysis, the malware sample is executed in a controlled environment while all network traffic is captured. Then the analyst checks what hosts the malware was communicating with and searches for any well-known network traffic patterns (e.g. spam sending). Network analysis is usually a great source of information about malware functionality and often helps to identify the particular malware family. Thanks to network analysis it is often possible to identify addresses of (command and control) (C&C) servers and specific botnet to which a malware belongs.
Advanced dynamic analysis (not covered in exercise)	During advanced dynamic analysis, malicious code is executed in a debugger – letting the analyst precisely follow the malware execution path. Debugger analysis reveals the various execution paths and algorithms used by malware (e.g. the encryption algorithm used for network communication). Advanced dynamic analysis is usually more time consuming than other types of analyses and requires good reverse engineering (RE) skills as well as a deep knowledge of system internals.
Advanced static analysis (not covered in exercise)	In advanced static analysis, malicious code is disassembled and then analysed for malware functionality and the algorithms used. Just as with advanced dynamic analysis, this type of analysis is usually time consuming and requires good reverse engineering (RE) skills and a deep knowledge of system internals. One advantage with static analysis is that malicious code is never executed. It is also possible to analyse parts of the code that are never executed during dynamic analysis. The disadvantage of this analysis in that it is usually hard to predict the execution path and follow registry and stack changes.
Automatic analysis	In automatic analysis, malware is uploaded to a dedicated system which will perform automatic preliminary analysis. Automatic analysis usually produces similar results to the basic static analysis, behavioural analysis and network analysis. It also usually takes least time of all other analyses and is often used to quickly check a malware sample.

2.3 Safety precautions

During the exercise students will analyse live samples of malicious code. To avoid accidental infection it is necessary to take proper precautions and follow safety rules throughout the exercise.

1. Samples should never be executed outside of the analysis environment and dedicated virtual machine (Winbox).
2. Binary samples shouldn't be copied to any external storage – this might cause accidental infection if not done properly.

3. When executing samples, make sure there is no direct access to the local network. At the beginning of the analysis it is a good practice to verify there is no Internet connectivity on the analysis virtual machine (VM).
4. After each analysis, the snapshot of the clean virtual machine (VM) should be restored (it is not necessary after automatic analyses). Before each analysis, verify that a clean snapshot was restored after the previous analysis.

2.4 Exercise remarks

Due to the nature of malware execution during exercise tasks that include live malware samples (behavioural analysis, network analysis and automatic analysis), some of the obtained results may not be identical to the results presented in this document. A certain level of randomness and unpredictability that often accompanies malware execution (for example malware creating files with random names or malware connecting to random IP addresses) leads to varying observed results. Furthermore in some situations even a small change of the operating system configuration or the current environment state may also affect malicious code behaviour. Despite this, obtained results should be still analogous to the results presented in this document – and can be analysed using the same techniques.

In case some malware sample doesn't execute on the student virtual machine or for some reason behaves completely different than described in this document, special offline results are provided in /home/enisa/enisa/ex3/results directory. Using these results, a student should be able to complete the greater part of the task without the need to execute malicious code.

3 Tools overview

This section presents list of tools used in this exercise. Some of the tools used in the exercise give similar results and can be used interchangeably (e.g. PEview and CFF Explorer). It is advised that students first try to run tools presented in this section in the clean system before using them in actual analysis.

3.1 Static analysis tools

- **PEiD** – popular tool allowing to detect and identify Portable Executable files. It detects if executable is packed with one of the popular packers or protectors. If the file is not packed it can identify what compiler was used to create the executable file. PEiD has also simple generic unpacking module.
  http://www.woodmann.com/collaborative/tools/index.php/PEiD
- **Exeinfo PE** – tool allowing to detect many popular packers, protectors and crypters. Additionally Exeinfo PE has a ripper module allowing to search executable files for embedded files in a few popular formats (PE, zip, rar, doc, image files, etc.).
  http://www.woodmann.com/collaborative/tools/index.php/ExeInfo_PE
- **PEview** – Portable Executable (PE) headers and Component Object File Format (COFF) viewer tool. Displays headers, directories, sections, import/export tables and resource information.
  http://wjradburn.com/software/
- **CFF Explorer** – Portable Executable (PE) headers viewer and editor. It is designed to make PE editing as easy as possible. Beside PE headers viewing and editing CFF Explorer contains integrated hex editor, simple disassembler and many other useful features.
  http://www.ntcore.com/exsuite.php

- **Resource Hacker** – popular tool to view, modify, rename, add, delete and extract resources in 32bit & 64bit Windows executables and resource files.
  http://www.angusj.com/resourcehacker/
- **BinText** – simple and powerful strings extractor tool. It extracts ASCII, Unicode and Resources strings from a binary file. BinText also enables you to set additional extraction criteria and strings filters based on string minimal and maximal length, allowed characters, etc. Extracted strings can be saved to a separate file.
  http://www.mcafee.com/us/downloads/free-tools/bintext.aspx
- **Upx** – one of the most popular executable packing tools. It allows to pack and unpack executable files.
  http://upx.sourceforge.net/

3.2 Dynamic analysis tools

- **Process Explorer** – powerful task manager and system monitor for Microsoft Windows. It provides the functionality of Windows Task Manager along with a rich set of features for collecting information about processes running on the user's system
  http://technet.microsoft.com/en-US/sysinternals/bb896653
- **Process Monitor** – tool from Windows Sysinternals suite. It monitors and displays in real-time all file system activity on a Microsoft Windows operating system. It combines two older tools, FileMon and RegMon and is used in system administration, computer forensics, and application debugging.
  http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx
- **Regshot** – open source tool allowing to quickly take snapshot of the registry and file system and then compare with the second one. Used to detect changes in system registry and file system (insertions, deletions, modifications).
  http://sourceforge.net/projects/regshot/
- **GMER** – application searching operating system for rootkit activity. It allows to detect hidden processes, hidden threads, hidden modules, hidden files, hooks on system functions and many more.
  http://www.gmer.net/

3.3 Network analysis tools

- **Tcpdump** – popular command-line network traffic sniffer and analyser. It allows to capture network traffic to the file in PCAP format.
  http://www.tcpdump.org/
- **Wireshark** – popular network traffic analyser, very similar to Tcpdump but with additional graphic user interface and integrated sorting, filtering and statistical options.
  https://www.wireshark.org/
- **Mitmproxy** – an interactive console program that allows to capture, inspect and edit HTTP/HTTPs traffic by acting as a transparent proxy.
  http://mitmproxy.org/
- **INetSim** – software suite used to simulate various network services in a lab environment.
  http://www.inetsim.org/

3.4 Automatic analysis tools

- **Cuckoo Sandbox** – open source automated malware analysis system. It allows to automatically execute suspicious files in a controlled and isolated environment in which it monitors malicious code activity. After analysis it creates a comprehensive malware analysis report.
  http://www.cuckoosandbox.org/

4 Task 1: Basic static analysis

In this task students will perform a basic static analysis of a binary sample. Static analysis is basically performed without running the malware as opposed to a dynamic analysis. A complete static analysis of a malware sample can be an extremely laborious process as it would require reverse engineering the source code and understanding its logic.

In this task the students will try to determine basic malware functionalities with a help from the trainer and the purpose of the task is to look for any special features which might be later used to create the file signature.

Basic static analysis covers the following topics:

- Determining file type and detecting packers or protectors.
- Strings extraction and analysis.
- Portable executable (PE) headers analysis.
- Import table analysis.
- Resources analysis.
- Scanning file for embedded objects (executables, images, etc.).

4.1 Sending sample to the analysis.

First restore the Winbox snapshot used for static and dynamic analyses (winbox-clean) as described in the exercise *Building artifact handling and analysis environment* (refer to this exercise on how to restore a snapshot if in doubt). When the snapshot is restored start the virtual machine.

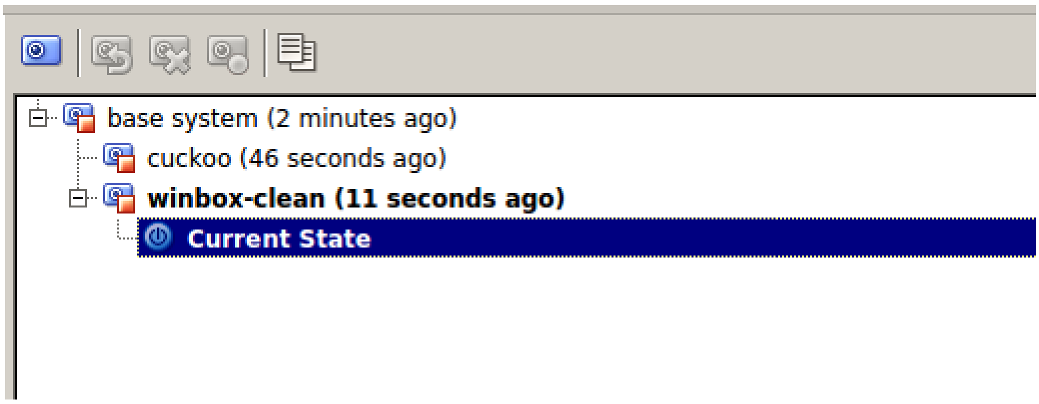


Figure 1. Restored winbox-clean snapshot.

Then, start Viper and find the aop.exe sample (screenshot) which should have been obtained as a result of the previous exercise (*Processing and storing artifacts*). In case there is no aop.exe sample it can be found in /home/enisa/enisa/ex3/samples directory from where it can be added to the Viper. Please refer to the exercise *Processing and storing artifacts* on how to use the Viper tool.

```
enisa viper > find name aop.exe
+---+---------+------------------------+-----------------------------------+------------------+
| # | Name    | Mime                   | MD5                               | Tags             |
+---+---------+------------------------+-----------------------------------+------------------+
| 1 | aop.exe | application/x-dosexec  | 7a0938b535f1bbd7a85065249bbbefd1  | enisa, bin, pe32 |
+---+---------+------------------------+-----------------------------------+------------------+
enisa viper > open -l 1
[*] Session opened on /opt/viper/projects/enisa/binaries/b/6/0/9/b60979f857ed87e348122d1288748a923c4f6a004474e8
8fa00bdf68964b1902
enisa viper aop.exe >
```

Figure 2. Finding aop.exe sample in Viper.

Then send the sample to Winbox using the previously created Viper module and exit Viper.

```
enisa viper aop.exe > lab-send
226 Successfully transferred "/sample/aop.exe"
enisa viper aop.exe >
```

Figure 3. Sending aop.exe sample to Winbox.

The sample should now appear in Winbox in c:\analyses\sample folder (refer to *Processing and storing artifacts* exercise to recall how the transfer works).

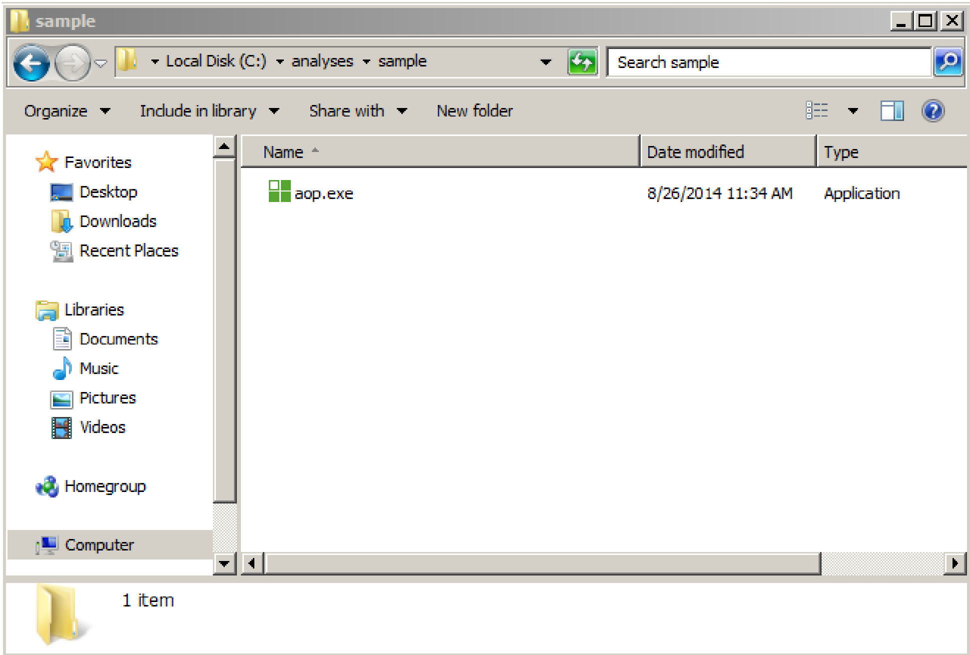


Figure 4. Malware sample after uploading to Winbox machine.

4.2 Detecting packers and protectors

Malware samples are often protected by so called packers and protectors[4]. Packers and protectors are dedicated tools intended to obfuscate and rewrite executable file structure in order to evade detection by antivirus (AV) products and hinder further analysis. Usually packed binary has a completely different structure than the original file. Moreover, protectors often add various protection functions such as virtualization detection, sandbox detection or debugger detection to executables.

In most cases a packed binary is very difficult for static analysis. Consequently a binary needs to be unpacked first; otherwise we can rely only on dynamic analysis findings. Unpacking a malware sample

[4] http://www.ecsl.cs.sunysb.edu/tr/TR237.pdf

isn't always a trivial task, often requiring good reverse engineering skills. Malware unpacking isn't the subject of this exercise.

There are two popular tools to detect packers signatures: PEiD and ExeInfoPE.

First open the aop.exe sample in PEiD:

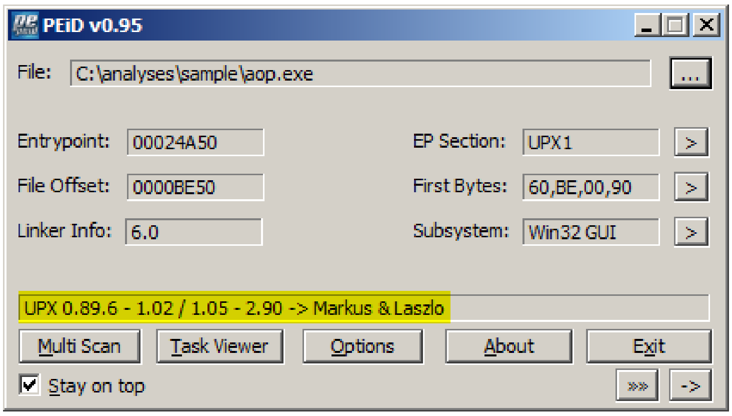


Figure 5. PEiD window - UPX packer detected.

It indicates (highlighted in yellow) that malware was most likely packed using UPX packer.

Verify PEiD findings and then open the sample in ExeInfo PE tool:

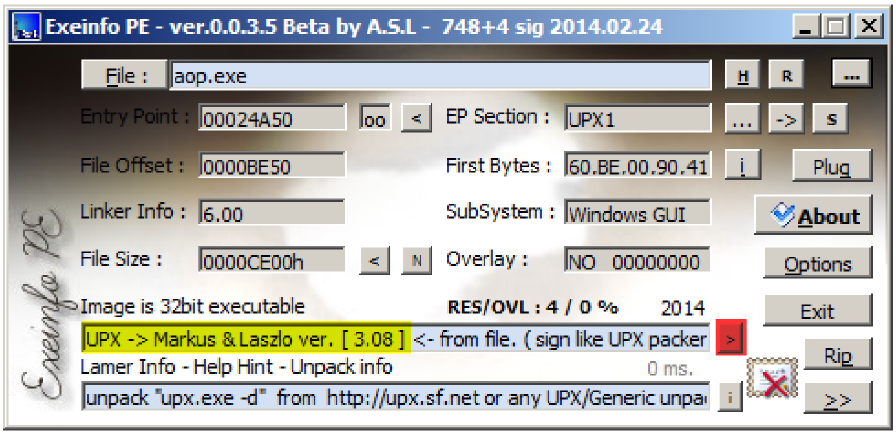
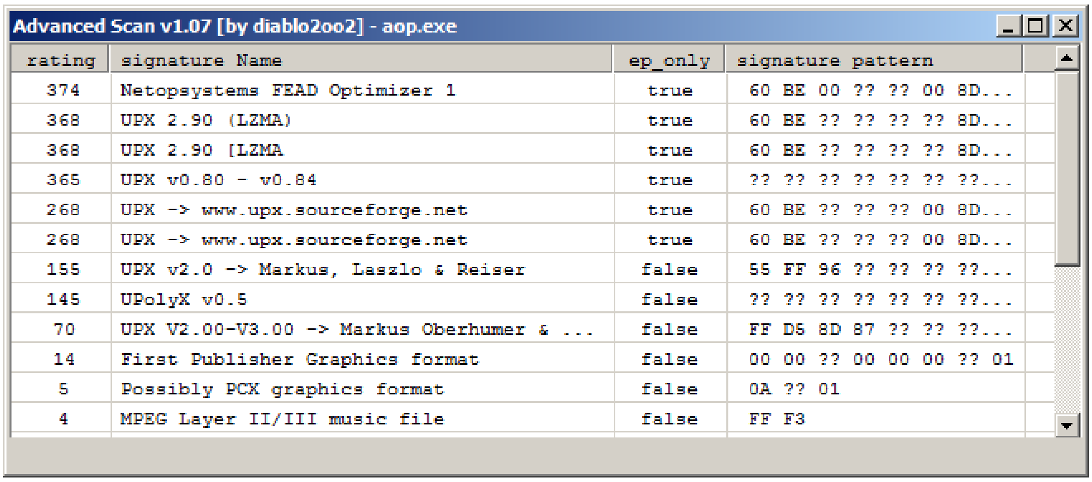


Figure 6. Exeinfo PE window - UPX packer detected.

ExeInfo PE confirms that the sample is most likely packed by UPX. You can also use advanced scan feature by clicking '>' button (highlighted in red). This will show other possible packers matching this particular file.



Advanced Scan v1.07 [by diablo2oo2] - aop.exe

rating	signature Name	ep_only	signature pattern
374	Netopsystems FEAD Optimizer 1	true	60 BE 00 ?? ?? 00 8D...
368	UPX 2.90 (LZMA)	true	60 BE ?? ?? ?? ?? 8D...
368	UPX 2.90 [LZMA	true	60 BE ?? ?? ?? ?? 8D...
365	UPX v0.80 - v0.84	true	?? ?? ?? ?? ?? ?? ??...
268	UPX -> www.upx.sourceforge.net	true	60 BE ?? ?? ?? 00 8D...
268	UPX -> www.upx.sourceforge.net	true	60 BE ?? ?? ?? 00 8D...
155	UPX v2.0 -> Markus, Laszlo & Reiser	false	55 FF 96 ?? ?? ?? ??...
145	UPolyX v0.5	false	?? ?? ?? ?? ?? ?? ??...
70	UPX V2.00-V3.00 -> Markus Oberhumer & ...	false	FF D5 8D 87 ?? ?? ??...
14	First Publisher Graphics format	false	00 00 ?? 00 00 00 ?? 01
5	Possibly PCX graphics format	false	0A ?? 01
4	MPEG Layer II/III music file	false	FF F3

Figure 7. Advanced scan feature in Exeinfo PE.

Fortunately UPX is a quite simple and easy to unpack packer (and also still quite often seen in the wild). To unpack aop.exe we will use the standard upx.exe utility available on the Winbox.

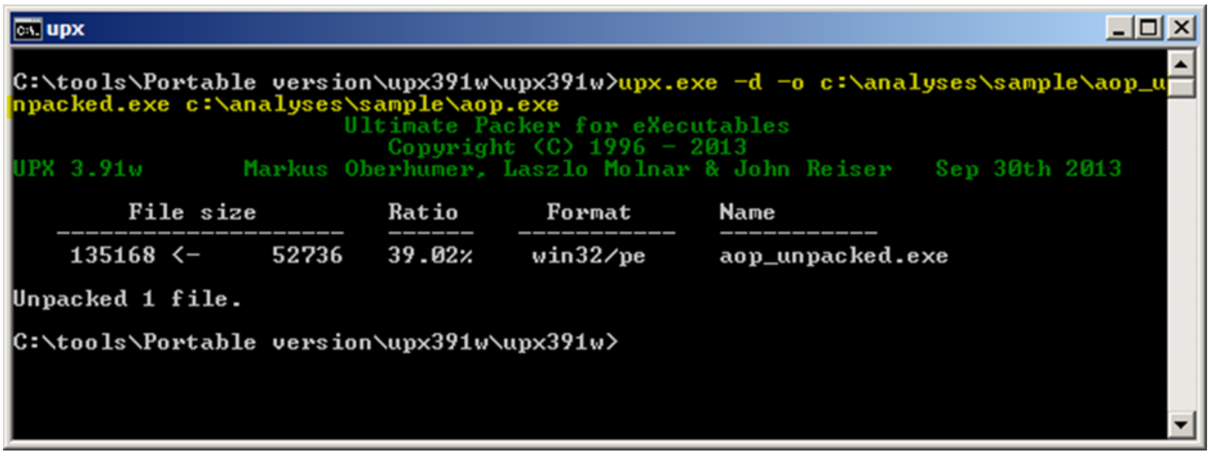


Figure 8. Unpacking malware sample with upx tool.

The unpacked sample should be saved as c:\analyses\sample\aop_unpacked.exe. To verify if it was successfully unpacked and is not protected by any other protector open it in PEiD.

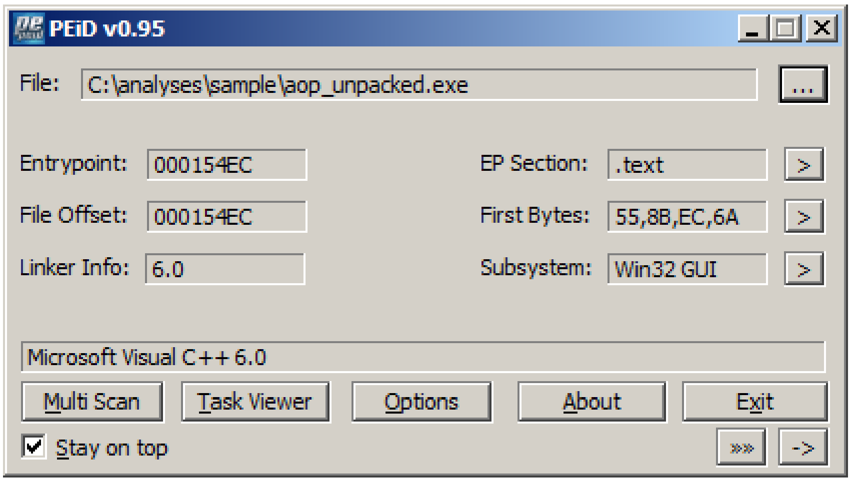


Figure 9. Checking unpacked sample in PEiD.

Based on new PEiD results we can assume that aop_unpacked.exe is not protected by another packer and was likely compiled using Microsoft Visual C++ version 6.0. (Microsoft Visual C++ and Microsoft Visual Studio are popular software development tools used widely by programmers all over the world)[5].

NB: File aop_unpacked.exe will be used instead of aop.exe in all following analyses.

4.3 Strings extraction and analysis

One very useful technique in malware analysis is string analysis. In many cases using strings obtained from the binary file we can reason about some of the features of the malicious code. For example if we find a list of SMTP servers we might suppose that malware might be sending spam messages.

To extract strings from the sample file (aop_unpacked.exe) use the BinText tool. This tool allows to extract all ASCII and Unicode strings from the binary file also allowing to apply certain filters on minimal string length and allowed characters.

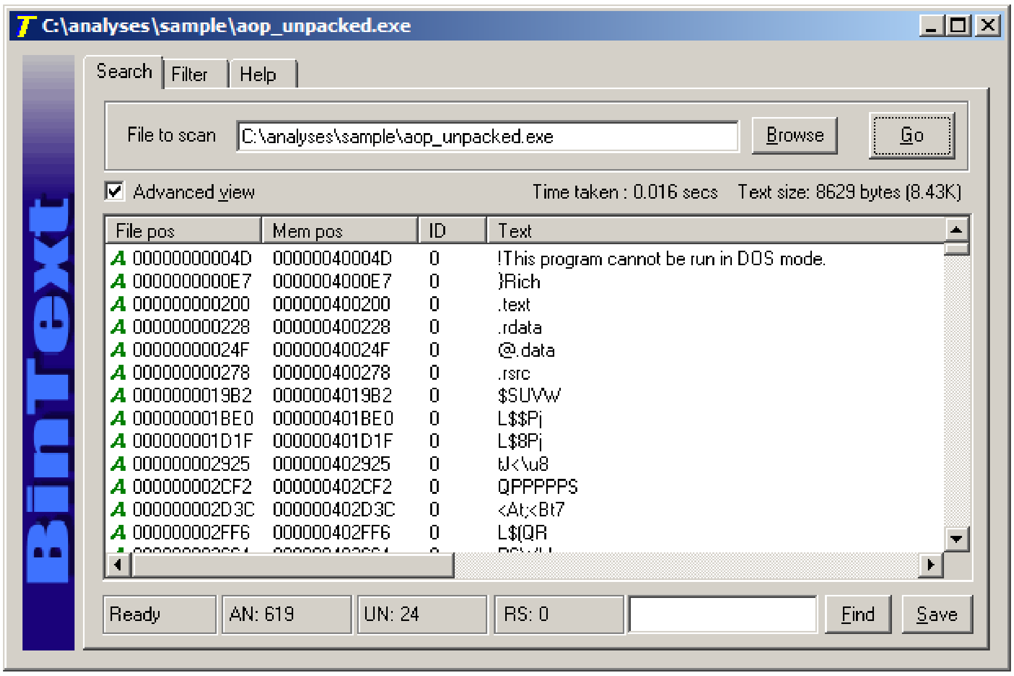


Figure 10. BinText window after opening unpacked sample file.

After extracting strings it is good to save them to the results directory for any further analyses.

[5] See http://www.visualstudio.com/ for more details on this development environment.

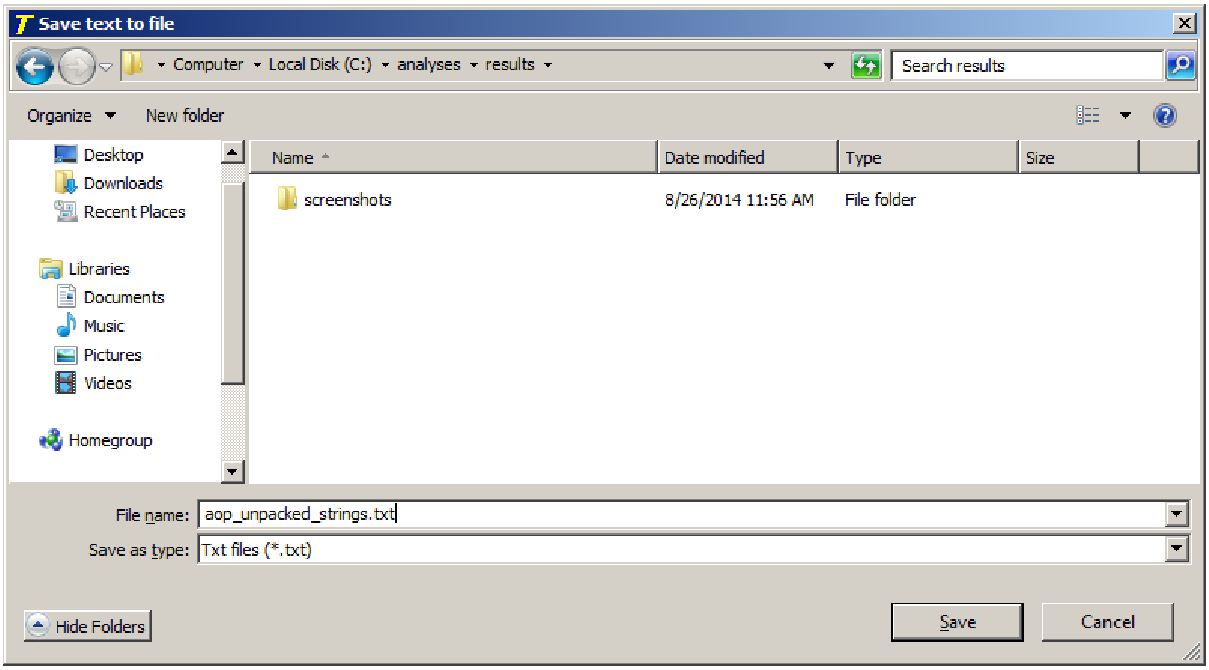


Figure 11. Saving strings extracted by BinText to a file.

Next scroll down the list of all discovered strings trying to find any useful information about malware and its functionality. Students should look for strings such as IP and URL addresses, names of commands, Windows function and libraries names, usernames, e-mails, headers of various protocols (IRC, HTTP, etc.) or any other unique and characteristic names.

A	0000000192F2	0000004192F2	0	MSVFW32.dll
A	0000000192FE	0000004192FE	0	SHELL32.dll
A	00000001930A	00000041930A	0	SHLWAPI.dll
A	000000019316	000000419316	0	USER32.dll
A	000000019321	000000419321	0	WININET.dll
A	00000001932D	00000041932D	0	WS2_32.dll
A	000000019338	000000419338	0	WTSAPI32.dll
A	000000019348	000000419348	0	lstrcpyA
A	000000019352	000000419352	0	SetEvent
A	00000001935C	00000041935C	0	InterlockedExchange
A	000000019372	000000419372	0	CancelIo
A	00000001937C	00000041937C	0	DeleteFileA
A	00000001938A	00000041938A	0	GetLastError
A	000000019398	000000419398	0	CreateDirectoryA
A	0000000193AA	0000004193AA	0	GetFileAttributesA
A	0000000193BE	0000004193BE	0	lstrlenA

Figure 12. Windows functions and library names.

Here we can see a fragment of the DLL names[6] and imported functions list. It is good to compare such a list with names found in import table in PE file (*Portable Executable*) – this will be covered in a later step. Sometimes malware dynamically loads certain libraries and functions making them not listed in the PE file import table.

[6] http://support.microsoft.com/kb/815065

A	00000001B12C	00000041B12C	0	%s*.*
A	00000001B134	00000041B134	0	FindFirstFileA
A	00000001B144	00000041B144	0	LocalReAlloc
A	00000001B154	00000041B154	0	FindNextFileA
A	00000001B164	00000041B164	0	%s\%s
A	00000001B16C	00000041B16C	0	%s%s%s
A	00000001B174	00000041B174	0	%s%s*.*

Figure 13. Some path matching expressions found in strings list.

Patterns like %s*.* and %s\%s suggest they might be used as arguments to some system function calls for path matching or file searching. Also presence of functions such as FindFirstFileA and FindNextFileA suggest that malware might be searching certain files on local disk.

A	00000001B180	00000041B180	0	system\cURRENTcONTROLsET\sERVICES\%s
A	00000001B1A8	00000041B1A8	0	OpenSCManagerA
A	00000001B1C0	00000041B1C0	0	system\cURRENTcONTROLsET\sERVICES\

Figure 14. Registry keys found in strings list.

Registry keys related to Windows Services. This might suggest that malware is using system service as a self-preservation technique (persistence mechanism).

A	00000001B380	00000041B380	0	%%%c%c%%%c%c
A	00000001B398	00000041B398	0	%d.%d.%d.%d
A	00000001B3A4	00000041B3A4	0	192.168.1.244

Figure 15. Suspicious IP address

Unusual IP address from private address space. It is hard to say what it is used for but it might be a good starting point for further analysis (either dynamic debugging or more advanced static analysis of disassembled code).

A	00000001B3F0	00000041B3F0	0	HTTP/1.1
A	00000001B3FC	00000041B3FC	0	Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-sh
A	00000001B4AC	00000041B4AC	0	Accept-Language: zh-cn
A	00000001B4C8	00000041B4C8	0	Accept-Encoding: gzip, deflate
A	00000001B4EA	00000041B4EA	0	User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
A	00000001B532	00000041B532	0	Host:
A	00000001B53A	00000041B53A	0	Connection: Close
A	00000001B558	00000041B558	0	Referer: http://
A	00000001B56C	00000041B56C	0	:80/http://
A	00000001B57A	00000041B57A	0	Host:
A	00000001B586	00000041B586	0	Cache-Control: no-cache
A	00000001B5A0	00000041B5A0	0	Host:
A	00000001B5A8	00000041B5A8	0	HTTP/1.1
A	00000001B5B6	00000041B5B6	0	Connection: Keep-Alive

Figure 16. HTTP headers found in strings list.

Typical HTTP headers suggest that malware is likely using HTTP or HTTPs communication – to contact Command & Control server or for some other purposes.

A	00000001B5D8	00000041B5D8	0	cmd /c ping 127.0.0.1 -n 1&del "%s"
A	00000001B5FC	00000041B5FC	0	%s\svchost.exe

Figure 17. Strings with batch command.

Typical batch command used for self-removal.

A	00000001B69C	00000041B69C	0	GetStartupInfoA
A	00000001B6B0	00000041B6B0	0	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
A	00000001B6F8	00000041B6F8	0	1107791273.f3322.org

Figure 18. String characteristic for base64 encoding.

Characteristic string (*ABCD…*) typically used in Base64[7] encoding functions.

A	00000001B6B0	00000041B6B0	0	ABCDEFGHIJKLMNOPQRSTUVWXYZabc
A	00000001B6F8	00000041B6F8	0	1107791273.f3322.org
A	00000001B70D	00000041B70D	0	

Figure 19. Suspicious domain name.

Suspicious domain name 1107791273.f3322.org. This might be a domain of C&C server – needs further inspection.

A	00000001B798	00000041B798	0	Prsionaljrq
A	00000001B7FC	00000041B7FC	0	Prsionyta Instruments Domain Service
A	00000001B87C	00000041B87C	0	Providesmid a domain server for NI security.

Figure 20. Unusual unique names found in strings list.

Unusual names: *prsionaljrq*, *prsionyta* and *providesmid*. Such unique names usually distinctly identify particular malware family. They might be used to name malware itself, create signature or to search more information about this particular malware on the web.

A	00000001BA70	00000041BA70	0	F-secure
A	00000001BA7C	00000041BA7C	0	f-secure.exe
A	00000001BA94	00000041BA94	0	FortiTray.exe
A	00000001BAA8	00000041BAA8	0	avg.exe
A	00000001BAB0	00000041BAB0	0	Norman
A	00000001BAB8	00000041BAB8	0	NVCSched.exe
A	00000001BAC8	00000041BAC8	0	ClamAV
A	00000001BAD0	00000041BAD0	0	agent.exe
A	00000001BADC	00000041BADC	0	Comodo
A	00000001BAE4	00000041BAE4	0	cfp.exe

Figure 21. List of AV products and process names.

AV product names among other strings suggest that this malware is likely trying to evade detection by disabling AV services.

A	00000001BF7C	00000041BF7C	0	asdfgh
A	00000001BF84	00000041BF84	0	1314520
A	00000001BF8C	00000041BF8C	0	5201314
A	00000001BF94	00000041BF94	0	caonima
A	00000001BF9C	00000041BF9C	0	88888
A	00000001BFA4	00000041BFA4	0	bbbbbb
A	00000001BFAC	00000041BFAC	0	12345678
A	00000001BFB8	00000041BFB8	0	memory
A	00000001BFC0	00000041BFC0	0	abc123
A	00000001BFC8	00000041BFC8	0	qwerty
A	00000001BFD0	00000041BFD0	0	123456
A	00000001BFDC	00000041BFDC	0	password
A	00000001BFE8	00000041BFE8	0	enter
A	00000001BFF8	00000041BFF8	0	xpuser
A	00000001C000	00000041C000	0	money

Figure 22. List of common usernames and passwords.

Common usernames and passwords. This means malware is probably performing some dictionary attacks.

[7] http://en.wikipedia.org/wiki/Base64

A	00000001C084	00000041C084	0	at \\%s %d:%d %s
A	00000001C098	00000041C098	0	F:\NewArea.exe
A	00000001C0A8	00000041C0A8	0	\\%s\F$\NewArea.exe
A	00000001C0BC	00000041C0BC	0	E:\NewArea.exe
A	00000001C0CC	00000041C0CC	0	\\%s\E$\NewArea.exe
A	00000001C0E0	00000041C0E0	0	D:\NewArea.exe
A	00000001C0F0	00000041C0F0	0	\\%s\D$\NewArea.exe
A	00000001C104	00000041C104	0	admin$\
A	00000001C10C	00000041C10C	0	\\%s\admin$\NewArea.exe
A	00000001C124	00000041C124	0	C:\NewArea.exe
A	00000001C134	00000041C134	0	\\%s\C$\NewArean.exe
A	00000001C14C	00000041C14C	0	\\%s\ipc$

Figure 23. Windows file sharing related strings.

Strings typical for Windows file sharing. This malware is probably using Windows file sharing services – for self-propagation or some other reasons.

Exercise:

1. **Extract list of strings from the packed binary file (aop.exe) and compare them to strings analysed in this step. What are the differences?**

In packed binary file there are much less meaningful strings. Most of the interesting strings found in this step aren't present on the strings list from the packed file or are split in smaller parts.

A	00000000A85C	00000042345C	0	Encodw
A	00000000A873	000000423473	0	-Agi:Mo
A	00000000A881	000000423481	0	/4.0 (0mpnbWk7
A	00000000A8CF	0000004234CF	0	http:/
A	00000000A908	000000423508	0	cmd /c p
A	00000000A92C	00000042352C	0	svfa.

Figure 24. Incomplete or split strings in the packed binary file.

4.4 PE structure and headers analysis

Windows executable file (PE) headers contain information about the executable file and how it should be executed. PE headers tell the operating system how it should load an executable file, what libraries are needed, where the beginning of the main routine code (code entry point) is or even when the binary file was created. During malware analysis it is worthwhile to analyse PE headers to search for any anomalies or indicators that the sample was packed (especially in case when unknown packer is used and standard packer detection tools will not help).

Open the sample in the PEview tool and switch to IMAGE_FILE_HEADER. One of the interesting fields in this section is *Time Date Stamp* which tells when the binary executable was likely linked. This field might have been intentionally tampered with but it doesn't happen often.

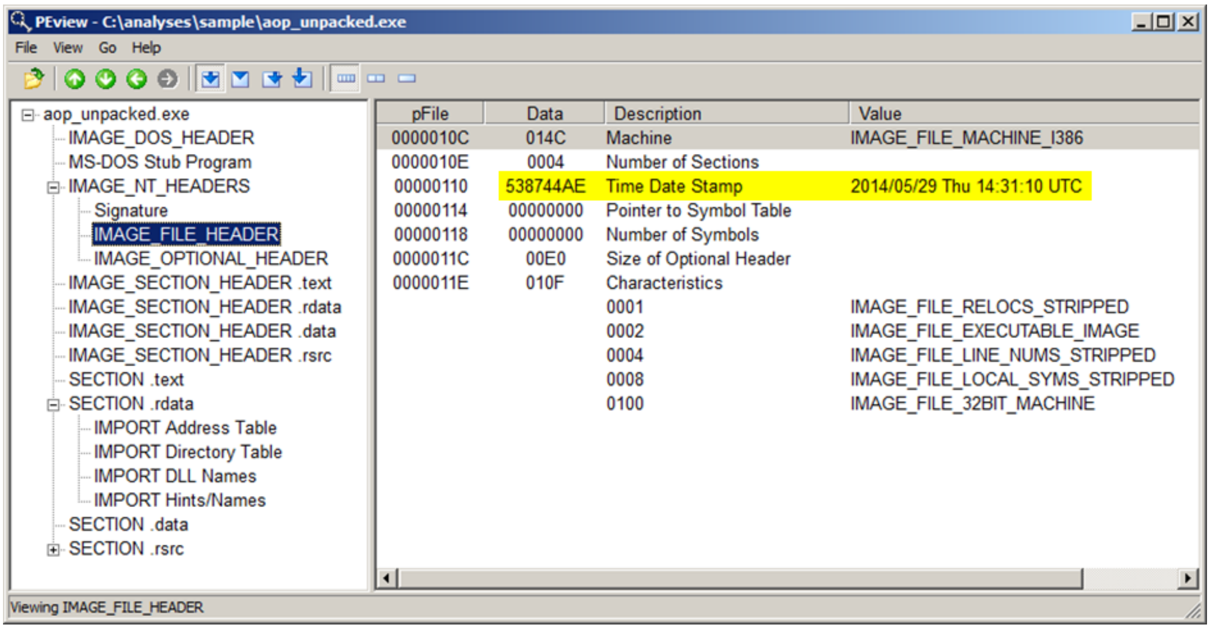


Figure 25. IMAGE_FILE_HEADER in PEview tool.

Next switch to IMAGE_OPTIONAL_HEADER and check the address of the entry point (EP). It will be used in the next step to determine in which PE section the entry point is located. In this case the entry point is located at the relative address 0x154EC.

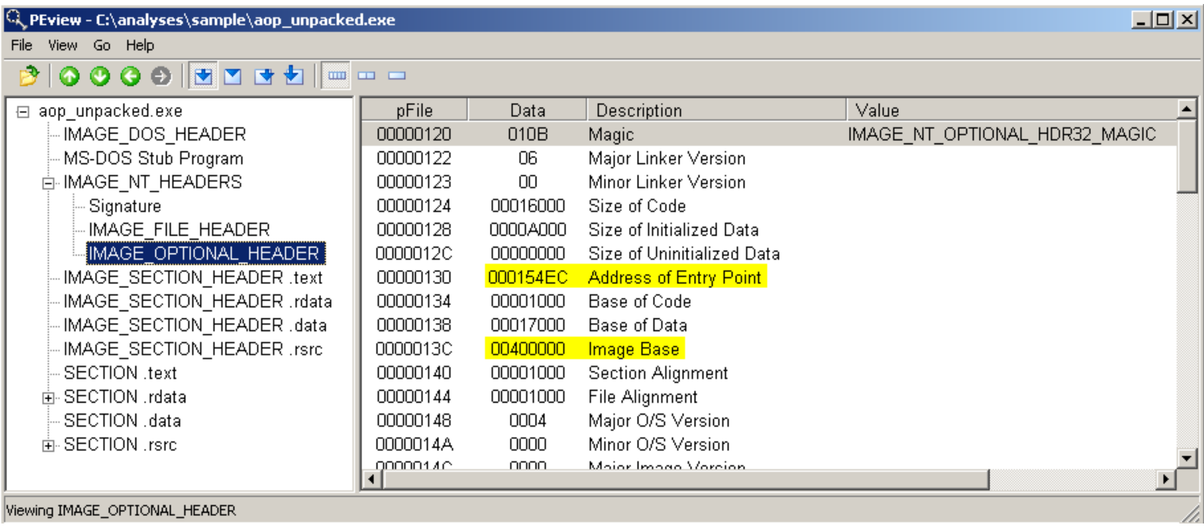


Figure 26. IMAGE_OPTIONAL_HEADER in PEview tool.

Then analyse PE file sections names and sizes. Based on PE section names it is sometimes possible to identify what packer or compiler was used to create the executable. For example UPX packed binaries typically have two sections named UPX0 and UPX1 while code compiled with Borland Delphi will typically have CODE, DATA, BSS, .rdata, .idata sections[8].

Then analyse the characteristics of the sections to check which of them appears to contain executable code (IMAGE_SCN_CNT_CODE, IMAGE_SCN_MEM_EXECUTE). Usually only one section should contain executable code (.text, CODE, etc.). Otherwise this indicates that some packer or protector was used. It is also good to compare the section size in memory with its raw size on disk. If the declared

[8] http://www.on-time.com/rtos-32-docs/rttarget-32/programming-manual/compiling/borland-delphi.htm

section size in memory is much greater than the section size on disk, then this also indicates that some packer or protector was most likely used.

Another indicator of a program being packed or somehow tampered with, is when a program entry point is located outside of the standard code section (.text, CODE, etc.). To check in which PE section the program entry point is located find the section for which RVA >= EP and EP <= RVA+Virtual Size (EP – previously checked address of entry point, RVA – section relative virtual address, Virtual Size – section size in memory). In this case, entry point 0x154EC is located in .text section because 0x1000 (.text RVA) <= 0x154EC (EP) <= 0x16345 (.text RVA+VirtualSize).

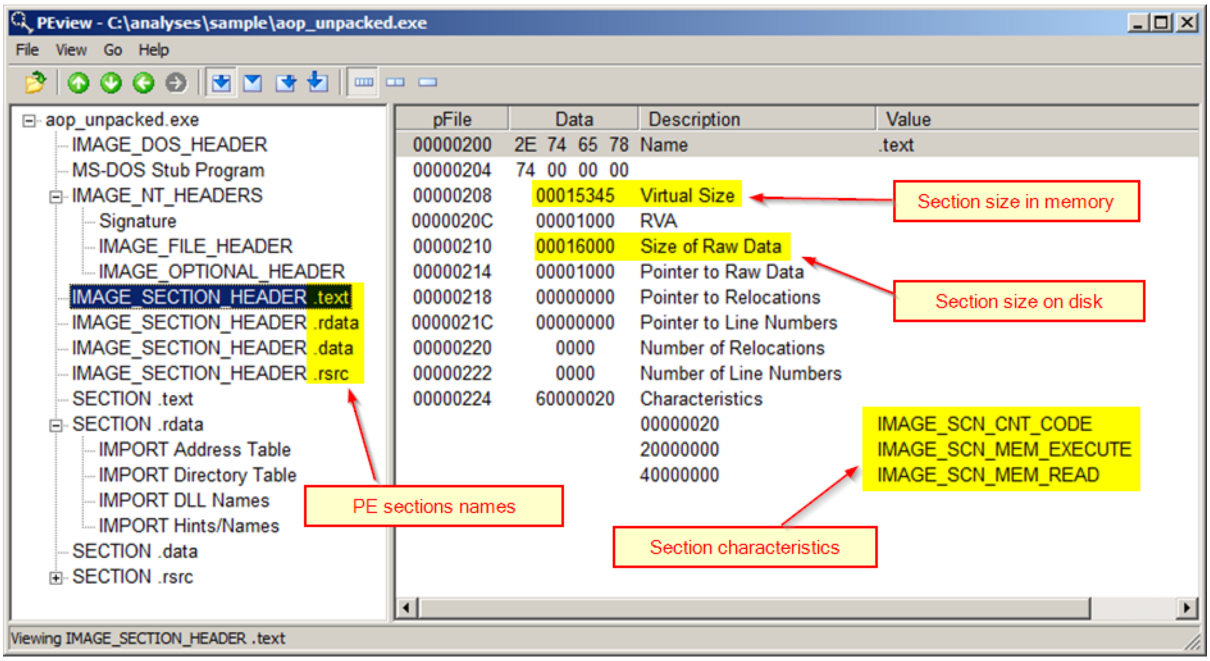


Figure 27. PE section view in PEview tool.

4.5 Import table analysis

Another important technique of static analysis is Import Address Table (IAT) analysis. By examining what functions and libraries the malware imports we can try to predict some of its functionality.

It is important to remember that IAT will not always contain all functions used by malicious code. Sometimes (especially in cases of packed or protected samples) the import table is shortened to only the most important functions, while the rest of the functions are imported dynamically during malware execution. In such a situation we need to use dynamic analysis techniques to determine the full set of functions used by the malware.

To analyse the Import Address Table we will use the CFF Explorer tool.

Open the sample file in CFF Explorer and switch to the *Import Directory* section. This section contains libraries imported by a malware sample. By clicking on each library name, a list of imported functions from this library opens in the bottom panel.

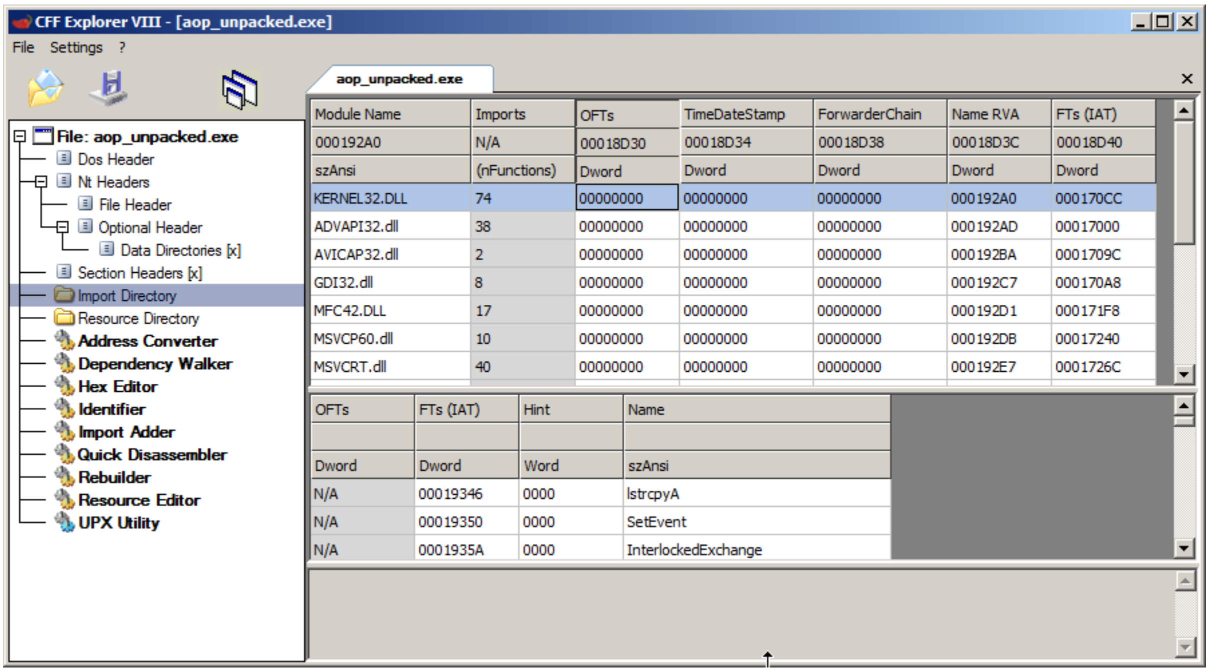


Figure 28. Import Directory view in CFF Explorer

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000192A0	N/A	00018D30	00018D34	00018D38	00018D3C	00018D40
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.DLL	74	00000000	00000000	00000000	000192A0	000170CC
ADVAPI32.dll	38	00000000	00000000	00000000	000192AD	00017000
AVICAP32.dll	2	00000000	00000000	00000000	000192BA	0001709C
GDI32.dll	8	00000000	00000000	00000000	000192C7	000170A8
MFC42.DLL	17	00000000	00000000	00000000	000192D1	000171F8
MSVCP60.dll	10	00000000	00000000	00000000	000192DB	00017240
MSVCRT.dll	40	00000000	00000000	00000000	000192E7	0001726C
MSVFW32.dll	7	00000000	00000000	00000000	000192F2	00017310
SHELL32.dll	1	00000000	00000000	00000000	000192FE	00017330
SHLWAPI.dll	1	00000000	00000000	00000000	0001930A	00017338
USER32.dll	39	00000000	00000000	00000000	00019316	00017340
WININET.dll	1	00000000	00000000	00000000	00019321	000173E0
WS2_32.dll	19	00000000	00000000	00000000	0001932D	000173E8
WTSAPI32.dll	2	00000000	00000000	00000000	00019338	00017438

Figure 29. List of imported libraries by aop_unpacked.exe

Here we see that the aop_unpacked.exe sample is importing functions from many different libraries. Among less common libraries are:

- Avicap32.dll – video capture functions
- Msvfw32.dll – bitmap/video compression and decompression functions
- Wtsapi32.dll – windows terminal services functions

We then analyze what functions are imported from each library, and search for functions that might point to some of the malware functionalities. Below is a list of a few more interesting functions.

N/A	0001A302	0000	WTSFreeMemory
N/A	0001A312	0000	WTSQuerySessionInformationA

Figure 30. Functions imported from wtsapi32.dll

As functions related to Windows Remote Desktop Service were detected, the malware might be trying to perform some operations in regard to the Remote Desktop Service. To get more information on how those functions are used, we would need to analyse the disassembled the code (advanced static analysis).

N/A	00019466	0000	SetLastError
N/A	00019474	0000	GetCurrentProcess
N/A	00019488	0000	CreateRemoteThread
N/A	0001949C	0000	WriteProcessMemory
N/A	000194B0	0000	VirtualAllocEx

Figure 31. Selected functions imported from kernel32.dll.

CreateRemoteThread and *WriteProcessMemory* functions are indicators that malware is injecting threads into other system processes. Most likely the intention is to hide its presence in the system or to tamper and interact with other processes (e.g. *information stealing*).

N/A	000195C4	0000	DisconnectNamedPipe
N/A	000195DA	0000	TerminateProcess
N/A	000195EC	0000	PeekNamedPipe

Figure 32. Selected functions imported from kernel32.dll.

TerminateProcess function suggest that malware might be trying to terminate some system processes. Knowing from the strings analysis, that the malware has hardcoded names of antivirus programs processes, we may guess that it will be trying to kill those processes to avoid detection.

N/A	000194F8	0000	TerminateThread
N/A	0001950A	0000	WinExec
N/A	00019514	0000	OutputDebugStringA

Figure 33. Selected functions imported from kernel32.dll.

The *WinExec* function suggests the malware might be trying to execute some system command.

N/A	000196A2	0000	Process32Next
N/A	000196B2	0000	Process32First
N/A	000196C2	0000	CreateToolhelp32Snapshot

Figure 34. Selected functions imported from kernel32.dll.

These functions are used to enumerate a process list. This confirms a previous suspicious that the malware might be trying to terminate certain processes or to inject remote threads to some of them.

N/A	000197C6	0000	RegOpenKeyA
N/A	000197D4	0000	RegQueryValueA
N/A	000197E4	0000	GetTokenInformation
N/A	000197FA	0000	LookupAccountSidA
N/A	0001980E	0000	CreateServiceA
N/A	0001981E	0000	RegDeleteKeyA
N/A	0001982E	0000	RegDeleteValueA
N/A	00019840	0000	RegEnumKeyExA

Figure 35. Selected functions imported from advapi32.dll.

These are functions used for registry operations. The malware is probably performing some registry operations. Also the presence of the function *CreateServiceA* suggests that the malware might create a system service – probably as a persistence mechanism.

N/A	00019A90	0000	capGetDriverDescriptionA
N/A	00019AAA	0000	capCreateCaptureWindowA

Figure 36. Functions imported from avicap32.dll.

These functions are used to create video capture. This suggests that the malware might have some spying functionality.

N/A	00019FCE	0000	ICSeqCompressFrameEnd
N/A	00019FE6	0000	ICCompressorFree
N/A	00019FF8	0000	ICClose
N/A	0001A002	0000	ICOpen
N/A	0001A00A	0000	ICSendMessage
N/A	0001A01A	0000	ICSeqCompressFrameStart
N/A	0001A034	0000	ICSeqCompressFrame

Figure 37. Functions imported from msvfw32.dll.

These video compression functions support the suspicion that the malware may try to capture a video sequence.

N/A	0001A144	0000	SetClipboardData
N/A	0001A156	0000	GetClipboardData
N/A	0001A168	0000	GetSystemMetrics

Figure 38. Selected functions imported from user32.dll.

These system clipboard functions suggest that the malware might be trying to monitor the system clipboard. It is another indicator of information stealing malware functionality.

N/A	0001A2F0	0000	InternetOpenUrlA

Figure 39. Function imported from wininet.dll.

InternetOpenUrlA function is used to retrieve data from FTP or HTTP location. Malware might be using this function to download additional configuration information from the Internet.

Exercise

1. **Analyse in CFF Explorer the Import Address Table of the packed binary file (aop.exe). What are the differences in comparison to the IAT of the unpacked sample?**

In the import address table (IAT) of the packed sample only six functions are imported from the kernel32.dll library and only one function from every other library. This is typical for UPX packed binaries.

aop.exe

Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
0000CB29	N/A	0000C980	0000C984	0000C988	0000C98C	0000C990
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
KERNEL32.DLL	6	00000000	00000000	00000000	00025B1C	00025A98
ADVAPI32.dll	1	00000000	00000000	00000000	00025B29	00025AB4
AVICAP32.dll	1	00000000	00000000	00000000	00025B36	00025ABC
GDI32.dll	1	00000000	00000000	00000000	00025B43	00025AC4
MFC42.DLL	1	00000000	00000000	00000000	00025B4D	00025ACC
MSVCP60.dll	1	00000000	00000000	00000000	00025B57	00025AD4
MSVCRT.dll	1	00000000	00000000	00000000	00025B63	00025ADC
MSVFW32.dll	1	00000000	00000000	00000000	00025B6E	00025AE4

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
N/A	00025C18	0000	FreeSid

Figure 40. IAT of the packed binary file.

4.6 PE resources analysis

Portable executable files usually contain an additional resources section which is used by the executable to store images, icons, dialog windows, menus or other data. Malware sometimes uses resource section store additional configuration data or files supposed to be dropped on a hard disk.

To examine the file resources section open the sample file in the Resource Hacker tool.

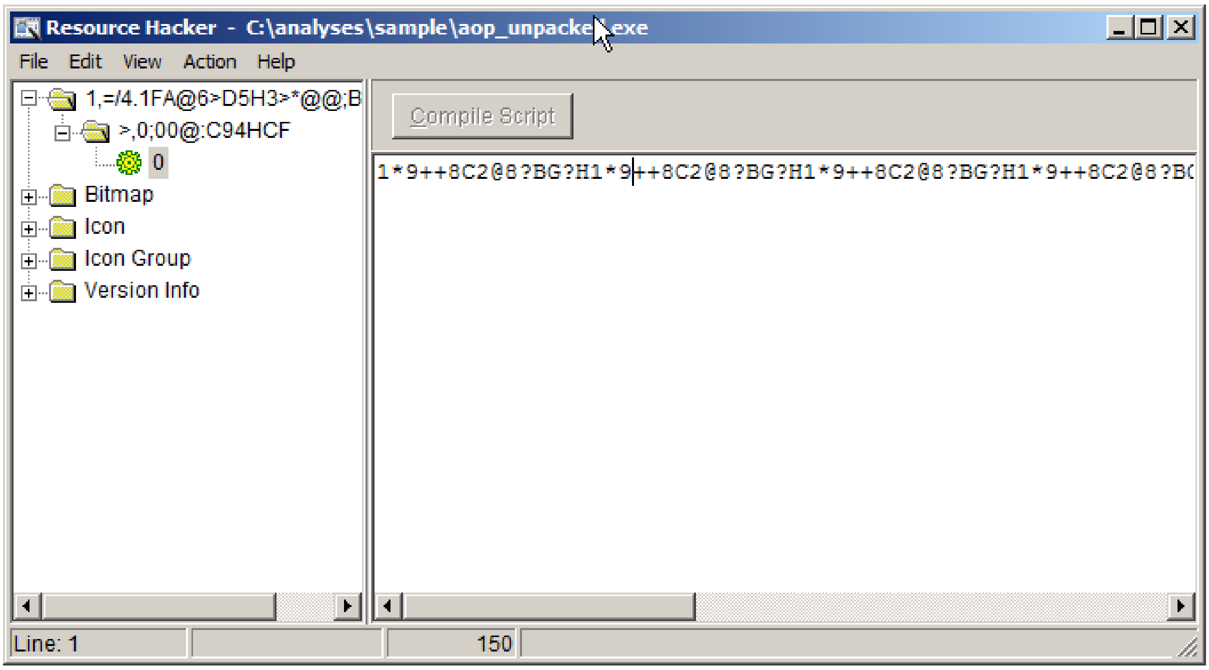


Figure 41. Suspicious resource in aop_unpacked.exe.

In this case we see that the malware sample contains a single suspicious resource. At this stage of the analysis it is hard to tell what it is used for. It might be some encrypted configuration string or just useless random data. To determine the role of this resource advanced dynamic or static analysis will probably be required.

Additionally the student might decide to export this resource to result files by right clicking on the resource and choosing the *Save* option.

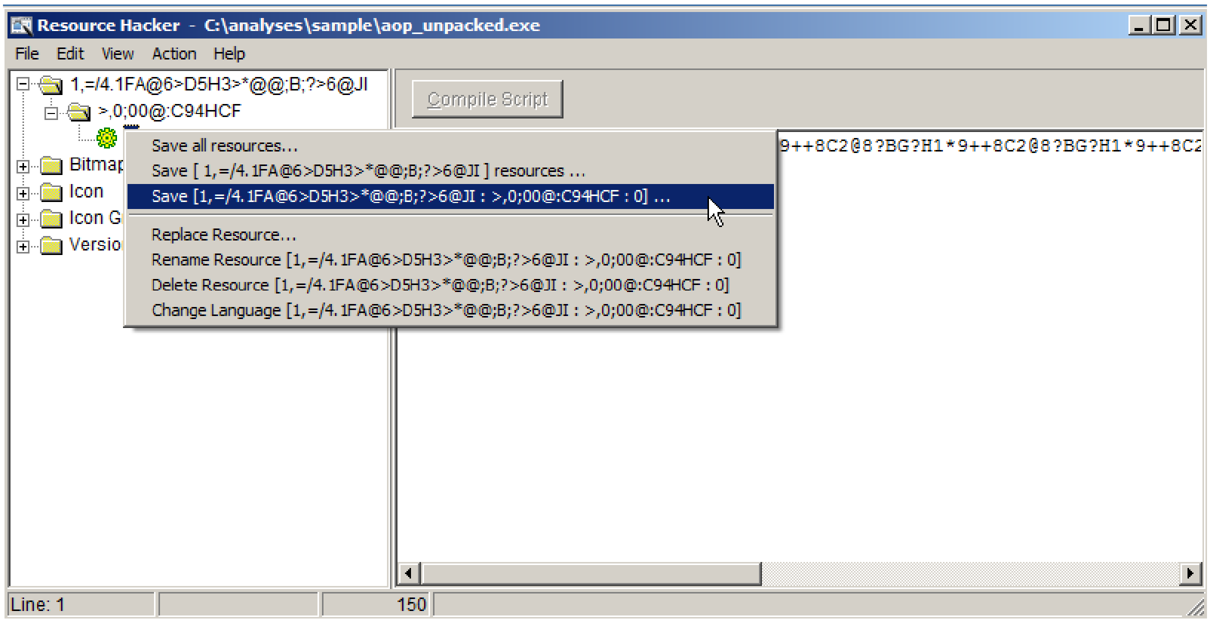


Figure 42. Exporting suspicious resource.

4.7 Searching for embedded objects

Malware might sometimes contain embedded objects outside of the resources section. Exeinfo PE tool has a special function allowing to scan any file for embedded objects in popular formats such as PE files, MSI files, Word documents, images, etc.

To scan the sample for embedded objects, open it in Exeinfo PE and open the Ripper menu by clicking on the Rip button. Then choose what type of object Exeinfo PE should search for, or choose the *I'm hungry for Ripping* option to search for all known file types.

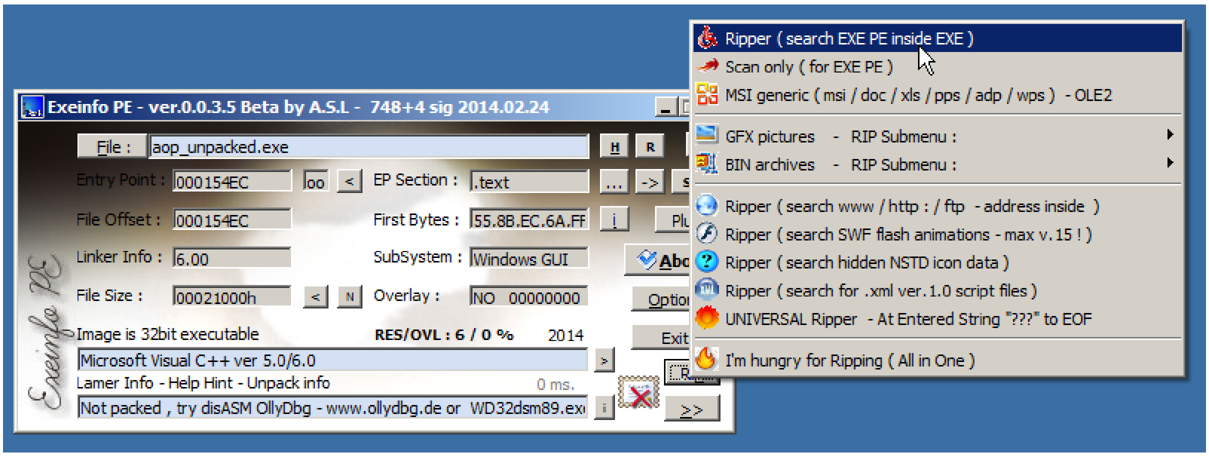


Figure 43. File ripping in Exeinfo PE.

If any embedded objects are be found they will be saved to the same directory in which the analysed sample resides. In the case of the aop_unpacked.exe binary sample, only two icon files were found.

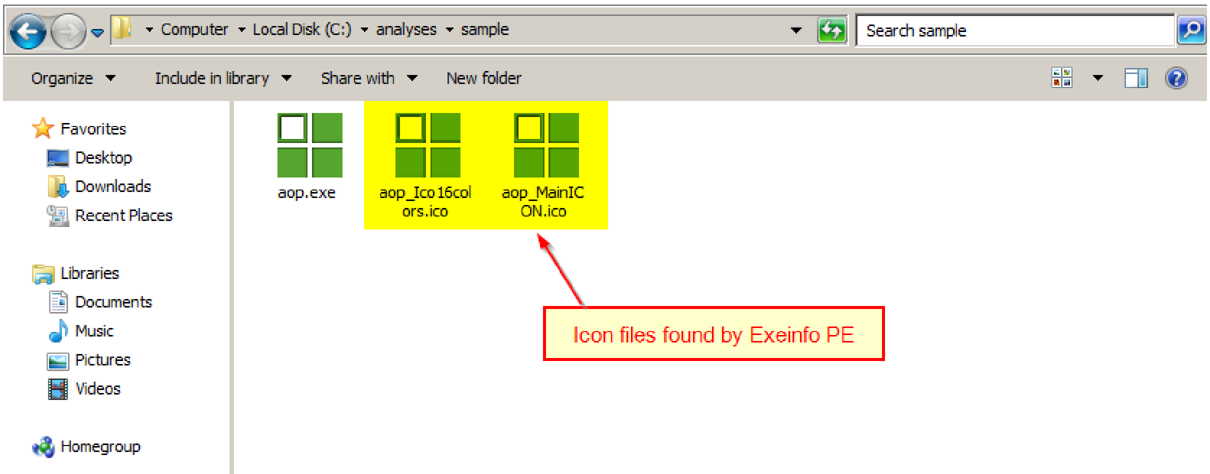


Figure 44. Icon files found by Exeinfo PE.

4.8 Finishing analysis

After the analysis is finished, copy and paste the obtained result files that you want to preserve into the directory C:\analyses\results.

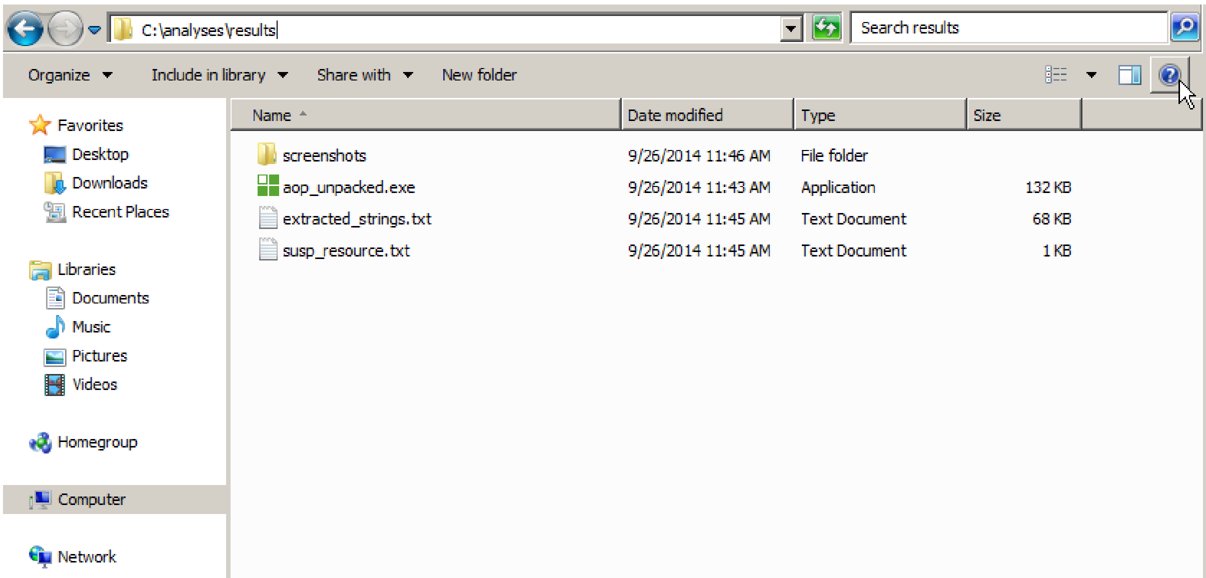


Figure 45. Analysis results in C:\analyses\results directory.

Then switch to Styx machine window and go to /lab/analyses directory and download the results in a separate subdirectory using the lab-get results script.

```
enisa@styx:~$ cd /lab/analyses/
enisa@styx:/lab/analyses$ mkdir aop.exe
enisa@styx:/lab/analyses$ cd aop.exe/
enisa@styx:/lab/analyses/aop.exe$ lab-getresults -d static_analysis
Storing results to: static_analysis
enisa@styx:/lab/analyses/aop.exe$ ls
static_analysis
enisa@styx:/lab/analyses/aop.exe$ ls static_analysis/
aop_unpacked.exe  extracted_strings.txt  screenshots  susp_resource.txt
enisa@styx:/lab/analyses/aop.exe$
```

Figure 46. Downloading results to Styx.

After the results are downloaded, shutdown Winbox machine and restore the clean snapshot.

4.9 Extra samples

As an extra exercise, students can analyse additional malware samples using the techniques learnt in this task. The extra samples names are: cutw231.exe, faktura.exe, svcost.exe. Samples can be found in /home/enisa/enisa/ex3/extra.

For each sample, it should be possible to point to some of the functionalities. After each analysis students should have an open discussion to share their findings.

5 Task 2: Behavioural analysis

In this task, the participant will execute malicious code in a virtual machine in order to observe what changes it will make to the operating system. Based on the observed changes, students will try to figure out how the malware works and what the indicators of the system infection are.

Behavioural analysis will cover following topics:

- Detecting new process creation

- Detecting file system and registry changes
- Detecting rootkit artifacts using Gmer
- Analysing in-memory strings
- Monitoring system events

5.1 Analysis remarks

In this task, live malware samples will be executed on the dedicated virtual machine. As previously mentioned, proper security precautions should be taken. All analyses will be done in the INetSim mode – preventing the malware from making any direct access to the external network.

After executing the malware sample in the VM, the user should keep in mind that malware (especially rootkits) sometimes change the operating system's behaviour to hide its presence. For example malware might hook file listing routines to hide its files on the file system.

Various tools used during the dynamic analysis might sometimes give false positive results (e.g. Gmer always detecting the same two suspicious changes). Consequently it is good to test the tools before executing actual malware to understand what the expected outcome might be.

During normal operating system operation there are many system processes and services running in the background. Those processes perform various tasks sometimes resulting in various changes in the operating system (e.g. creating pre-fetched files for executed binary files). This is particularly the case with newer operating systems versions like for example Windows 7 or 8. Those changes shouldn't be mistaken with the changes done by malware.

5.2 Preparing analysis

First restore a clean snapshot of the Winbox VM and make sure that current network mode is set to INetSim network simulator.

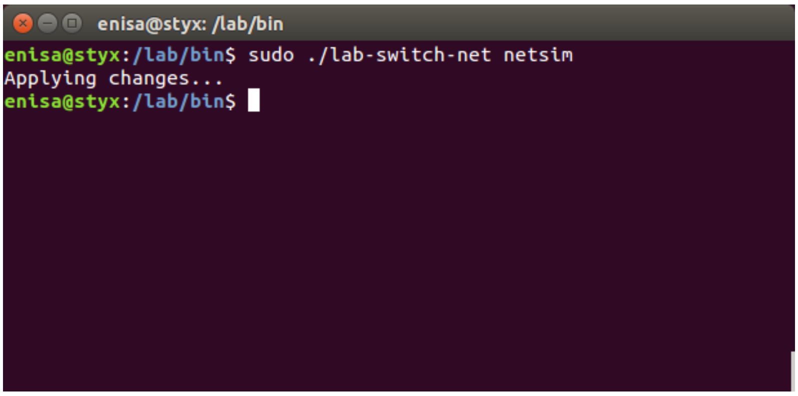


Figure 47. Switching network mode to network simulator

Then using Viper and the previously created malware collection, send the sample named 1102231642.exe to the Winbox machine. If there is no such sample in Viper, it can be copied from the directory */home/enisa/enisa/ex3/samples*.

```
enisa@styx: /opt/viper
enisa viper > find name 110223*
+---+----------------+-----------------------+----------------------------------+------+
| # | Name           | Mime                  | MD5                              | Tags |
+---+----------------+-----------------------+----------------------------------+------+
| 1 | 1102231642.exe | application/x-dosexec | 9482a68dd9ce5b2feda4ac139d8d12cf |      |
+---+----------------+-----------------------+----------------------------------+------+
enisa viper > open -l 1
[*] Session opened on /opt/viper/projects/enisa/binaries/d/c/e/c/dcec3a15d840be299f92c7d8c6330b75a9ae9718588b74
1a0b9c1c5756787d9c
enisa viper 1102231642.exe >
```

Figure 48. Finding sample in Viper

```
enisa@styx: /opt/viper
enisa viper 1102231642.exe > lab-send
226 Successfully transferred "/sample/1102231642.exe"
enisa viper 1102231642.exe >
```

Figure 49. Sending sample to the Winbox machine.

Next, switch to the Winbox window and start the following tools: Process Explorer, Process Monitor and Regshot. Refer to *Building artifact handling and analysis environment* exercise for the descriptions of these tools.

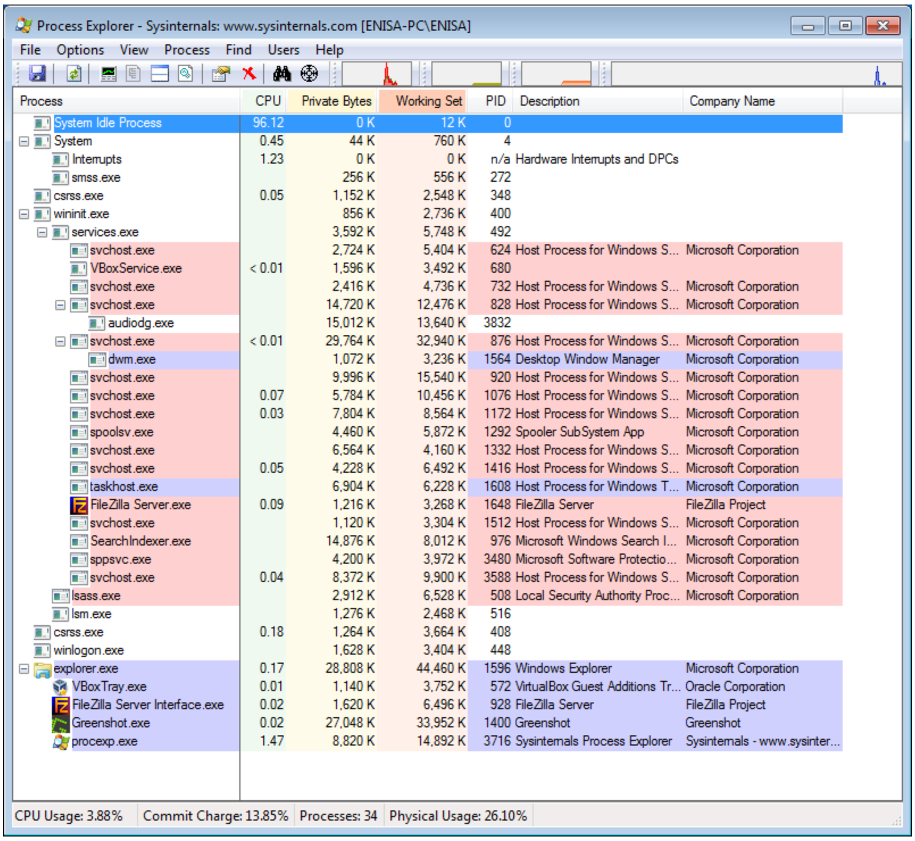


Figure 50. Process Explorer window.

After starting Process Monitor disable capturing events and clear capture view.

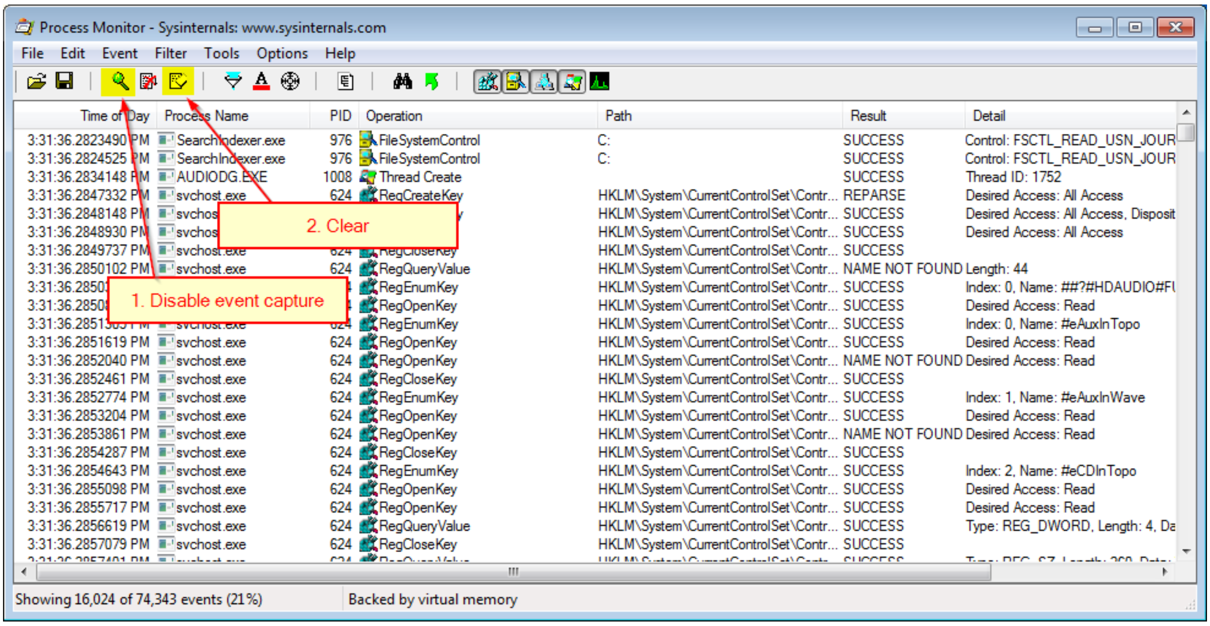


Figure 51. Disabling event capture and clearing Process Monitor

After starting Regshot check “*Scan dir*” option and set it to C:\.

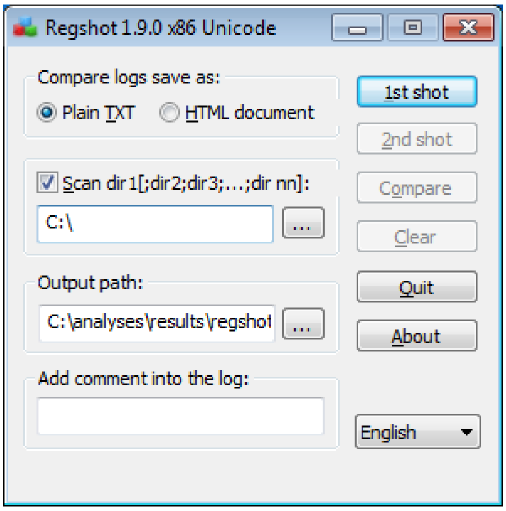


Figure 52. Regshot window.

Now the analysis environment is ready for the basic behavioural analysis. At this point the student might consider creating an additional snapshot just before executing the malware sample. If anything goes wrong during the analysis, or the student is uncertain about some specific malware behaviour, he could then use this snapshot to quickly restore the VM to the clean state with all of the tools already running and the with the malware sample already uploaded.

This snapshot should be distinctively named so it wouldn't be missed in the future and accidently merged with clean snapshot.

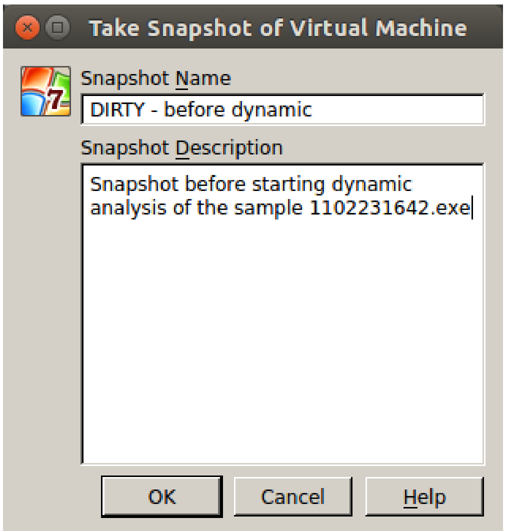


Figure 53. Creating snapshot before executing malware sample.

If the student decides to restore the snapshot, Winbox will be restored to its previous state. In particular all files in C:\analyses\results will be overwritten. If there are already some meaningful results stored in this directory, the student should consider downloading them with *lab-getresults* tool prior to restoring snapshot.

In case of any problems, an alternate way of finishing this task is to start only one tool at a time instead of starting all tools in a single analysis.

1. Start next single tool (Regshot, Process Explorer, Process Monitor, etc.).
2. Execute malware sample.
3. Analyse results.

4. If there are any result files send them to Styx VM (*lab-getresults*).
5. Restore snapshot and go to 1.

This approach is slightly more time consuming but in specific cases might be a better solution.

In case the malicious sample is not executing on the student's virtual machine, use the offline results provided in /home/enisa/enisa/results/dyn1 directory. To use the offline results it is best to send entire dyn1 directory to the Winbox virtual machine.

Sending offline results to the VM

```
$ lab-sendfile /home/enisa/enisa/ex3/results/dyn1
```

5.3 Executing malware sample

First use the Regshot tool to create an image of the clean system before executing malware sample.

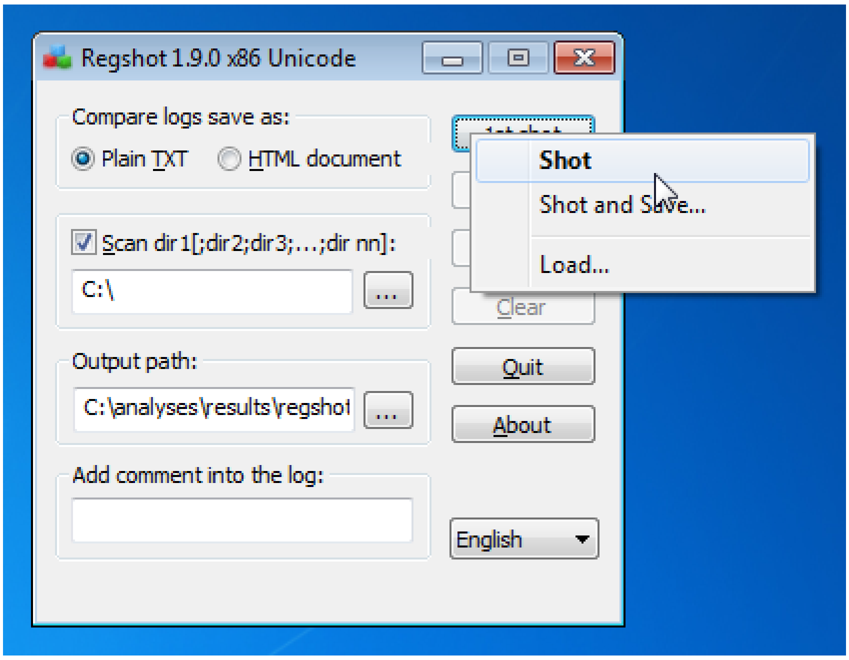


Figure 54. Taking first shot in Regshot

After Regshot finishes with the analysis (*2nd shot* button becomes active) start event capturing in Process Monitor.

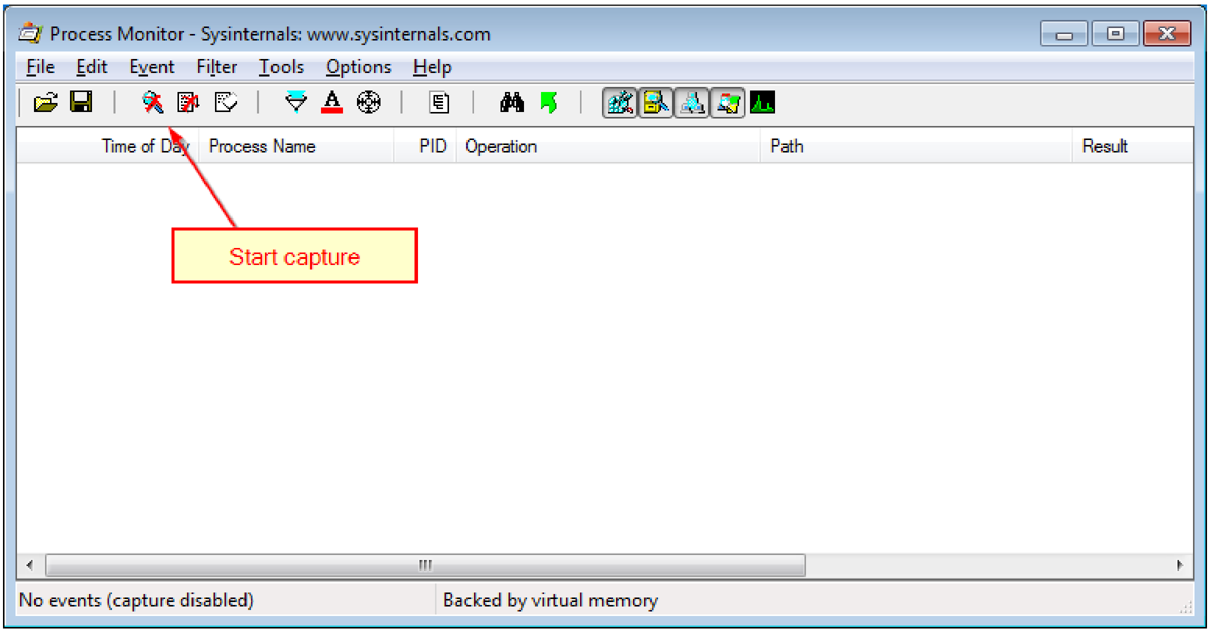


Figure 55. Starting event capture in Process Monitor.

Then student can execute the malware sample. At the same time, the student should pay attention to the Process Explorer window and observe if there are any changes on the process list.

After the malware sample is executed, the student should wait (up to a minute) until the malware is fully loaded in the system and finishes its installation routines. Then the student should stop the event capture in Process Monitor and then take a second shot in Regshot. This should be done before any further analysis in order to minimize the count of unimportant changes reported by Regshot and Process Monitor – being a result of a normal system activity and not malicious operations.

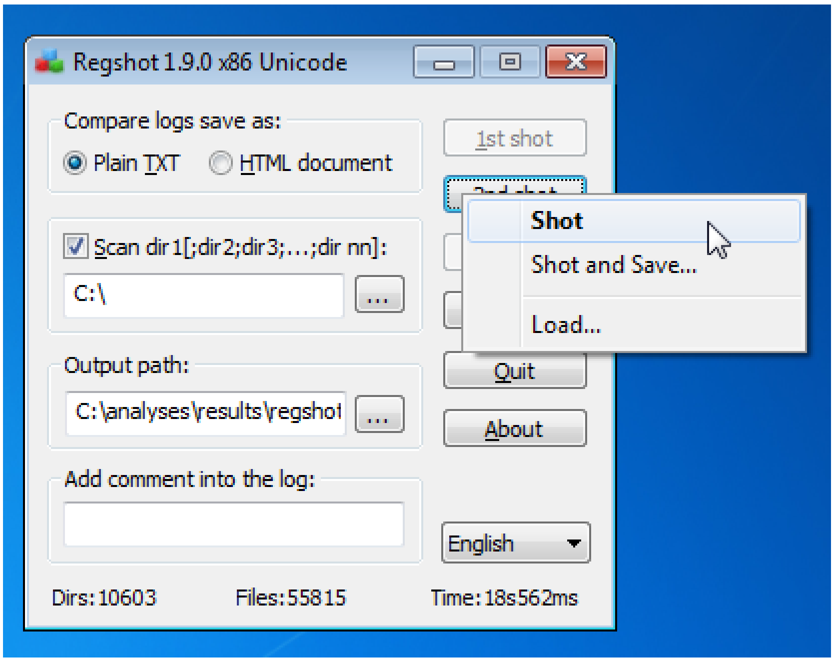


Figure 56. Taking second shot in Regshot

5.4 Process Explorer analysis

After executing the malware sample, new process 1102231642.exe almost instantaneously appears in the process list.

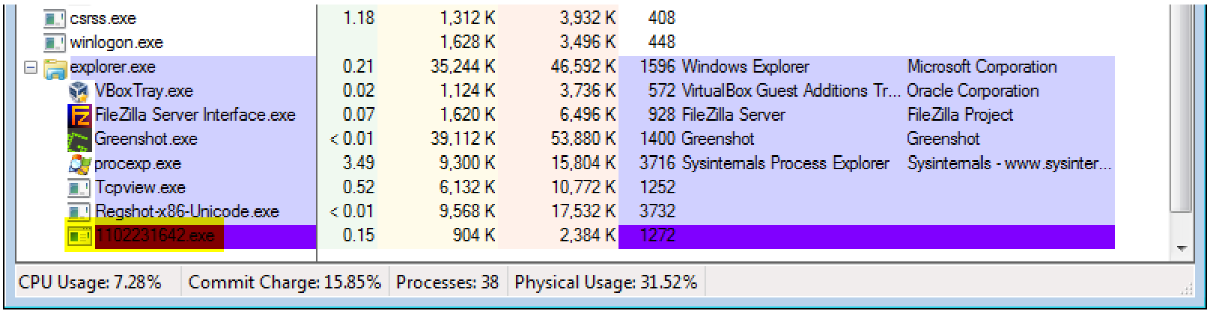


Figure 57. New malware process.

Process Explorer uses a distinct colour scheme to highlight various processes[9]. By default blue colour indicates that process is running in the same security context as Process Explorer. Pink colour indicates that process is hosting one or more Windows services. Purple means that process image has been most likely packed or compressed. Green and red colours points to new processes or the ones, that just exited.

Soon after the main malware process starts, it spawns four child processes: *win32.exe, explorer.exe, debug.exe, sysedit.exe* (random names, different in each analysis). Names of child processes suggests that those might be some system processes – which is one of the techniques sometimes used by malware to mislead system user. After spawning child processes malware process quits (red colour).

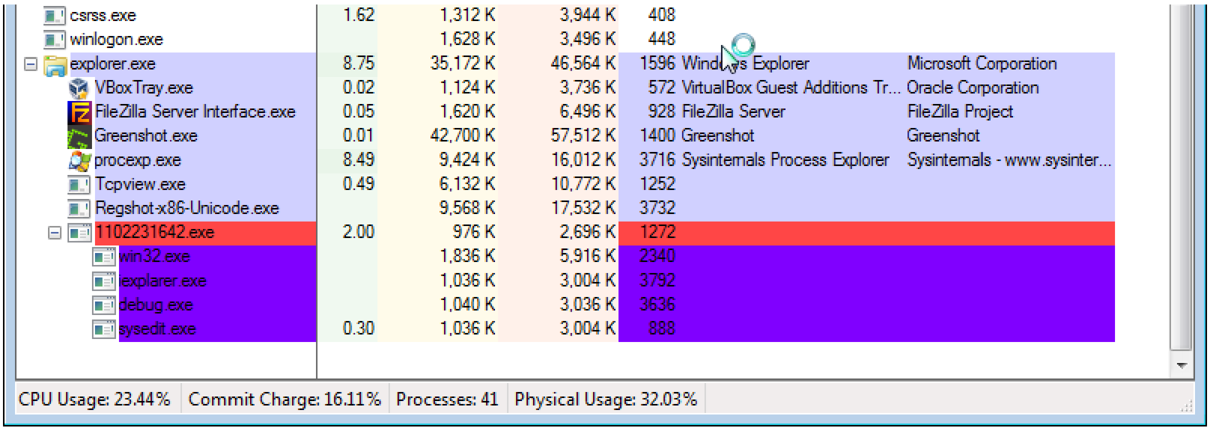


Figure 58. Malware process spawning child processes.

[9] http://www.microsoft.com/security/sir/strategy/default.aspx#!malwarecleaning_explorer

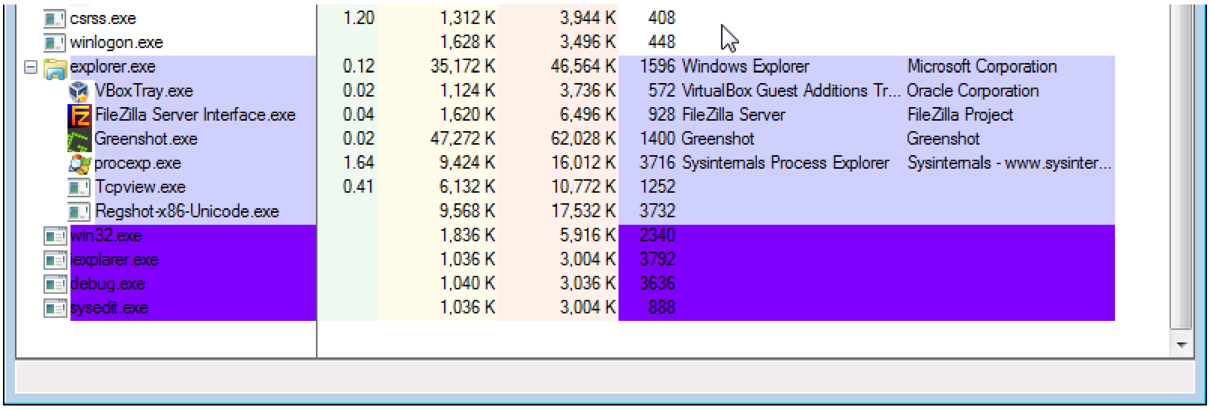


Figure 59. Child processes after main malware process quits.

Next students should further inspect all new processes by right clicking on them and opening the properties window. In the properties window, students can obtain various information about the process, such as image location, security context, performance data, list of threads, TCP/IP connections, as well as strings list. In this example we will examine the win32.exe process. Note that process names might be different during the analysis – then examine first new process on the list (analysis should be analogical).

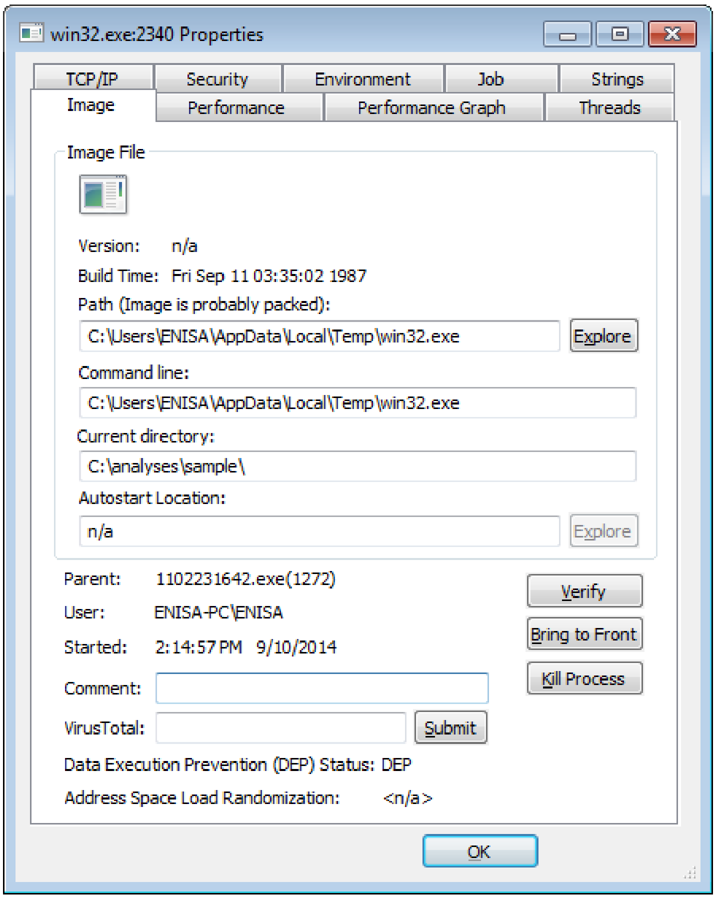


Figure 60. Process properties window.

In this case we see that images of suspicious child processes were stored in *%LOCALAPPDATA%\Temp* (C:\Users\ENISA\AppData\Local\Temp) directory which is typical location where malicious executables store their copies or drop other malware files.

Then students should switch to the *Strings* tab where they can inspect strings found in the process memory. Other means to achieve this goal would be to dump the process to a file and then use normal string analysis or attach to the process with a debugger and use the debugger to find all referenced text strings.

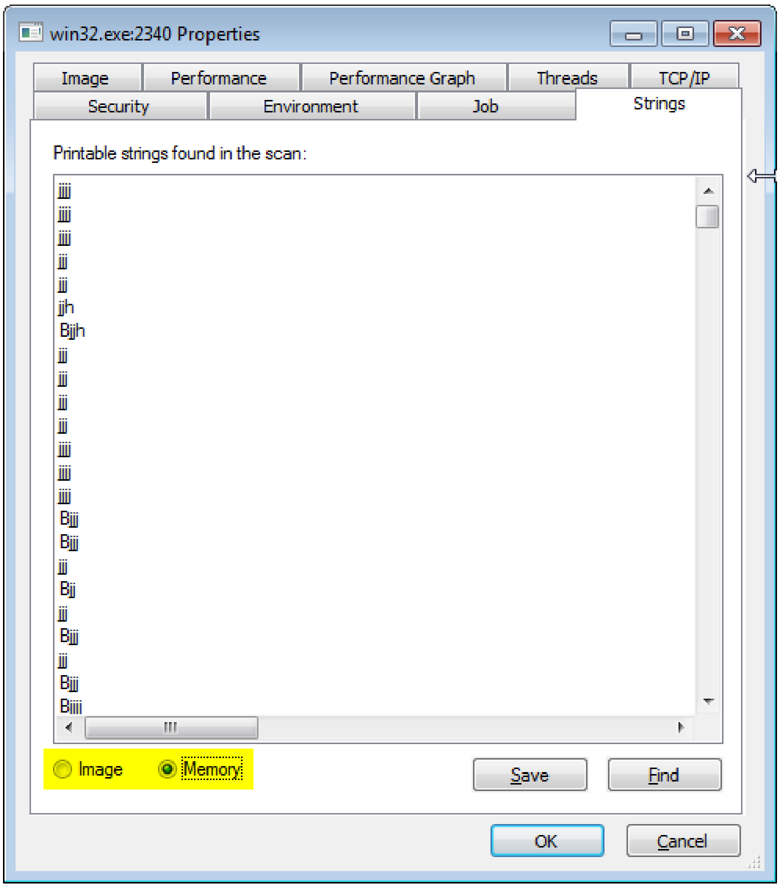


Figure 61. List of strings found in process memory.

Students should then compare the strings found in memory with the strings present in the image file (a simple visual comparison). Students should try to answer the following questions:

1. Do the strings found in memory differ from the strings obtained from the file (1102231642.exe)?
2. Are there any interesting strings in memory pointing to the malware's functionality or behaviour? (analysis similar to string analysis from previous task)?
3. Do strings found in memory differ for each child process (win32.exe, explorer.exe, etc.)?

In case of this malware sample, strings found in memory differ from strings found in the image. There are various strings pointing to potential malware functionality.

GetDC
GetClipboardData
GetClassNameA
FindWindowExA
IsWindow
ExitWindowsEx
GetWindowThreadProcessId
EmptyClipboard
SendMessageTimeoutA
SetForegroundWindow
SetWindowTextA
OpenClipboard
PostMessageA
EnumChildWindows
IsClipboardFormatAvailable

Figure 62. Names of Windows functions likely used by malware.

This list of WinAPI functions are most likely dynamically imported by the malware during execution. Those functions aren't present in either the executable image import table or in the strings found in image file.

perscrt.com
http://perscrt.com/rz/mn.php?ver=H1
im/pst.php
rz/report.php
Mozilla/4.0 (SPGK)

Figure 63. Suspicious url found in the strings list.

The suspicious URL with some PHP file names and a likely user-agent string. This suggests that the malware might be using http communication and this might be the address of the C&C server.

Below are images of some other distinctive groups of strings. Role of those strings isn't clear at this point of the analysis but they might be useful in later analyses.

continue shopping
download
submit
click here
sign in
register
log in
start
check out
cart
\drivers\etc\hosts
.exe

Figure 64. Suspicious strings and hosts file path.

```
7search.com
CHERR_
LURL
LC%ld-
LN%ld-
CURL
testovaya hren
fraud
cheat
img.php?
%ld_%ld
NOIE_
USEIE_
BODY_
text=
&url=
POST
%ld_%ld_%ld
%lX.ttp
Arial
%lX.png
POST
```

Figure 65. Group of other suspicious strings.

```
txtPTR
btnSubmit
http://%s/pic/sese.php?h=%s&q=%s
 Send
```

Figure 66. Some URL formatting string that might be used in communication with C&C server.

5.5 Regshot analysis

After completing the second shot in Regshot tool, students should click the *Compare* button to detect filesystem and registry changes between first and second shot. As a result a notepad window should appear with seven sections:

- Keys added (registry)
- Values deleted (registry)
- Values added (registry)
- Values modified (registry)
- Files added (file system)
- Files deleted (file system)
- Files [attributes?] modified (file system)

It is important to remember that Regshot uses standard system functions to detect any file system or registry changes. Consequently if malware alters those functions (e.g. to not list certain files), certain file system or registry changes may not be detected by Regshot. In most cases this applies to hiding malware files from the user. Such files can be often still be detected using results from other tools.

In the *Values added* section we see that the malware achieves persistence by adding new value *hsfio38fiosfh398rfisjhkdsfd* "C:\Users\ENISA\AppData\Local\Temp\win32.exe" in HKU\S-1-5-21-606041777-3127973734-2451401058-1001\Software\Microsoft\Windows\CurrentVersion\Run\. This is popular persistence mechanism used by malware letting it to be executed after each reboot.

```
----------------------------------
```

```
Values added: 15
----------------------------------
…
HKU\S-1-5-21-606041777-3127973734-2451401058-
1001\Software\Microsoft\Windows\CurrentVersion\Run\hsfio38fiosfh398rfisjh
kdsfd: "C:\Users\ENISA\AppData\Local\Temp\mdm.exe"
```

In the *Values modified* section we can see that the malware changed the values of Hidden and HiddenFileExt, which makes the operating system hide well known file extensions and disable showing hidden files.

```
----------------------------------
Values modified: 19
----------------------------------
HKU\S-1-5-21-606041777-3127973734-2451401058-
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden:
0x00000001
HKU\S-1-5-21-606041777-3127973734-2451401058-
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden:
0x00000000
HKU\S-1-5-21-606041777-3127973734-2451401058-
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt:
0x00000000
HKU\S-1-5-21-606041777-3127973734-2451401058-
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt:
0x00000001
…
```

In the *Files added* section we see that the malware added four executable files and one file with a .tmp extension.

```
----------------------------------
Files added: 10
----------------------------------
C:\Users\ENISA\AppData\Local\Temp\win32.exe
C:\Users\ENISA\AppData\Local\Temp\skaioejiesfjoee.tmp
C:\Users\ENISA\AppData\Local\Temp\explarer.exe
C:\Users\ENISA\AppData\Local\Temp\debug.exe
C:\Users\ENISA\AppData\Local\Temp\sysedit.exe
C:\Windows\Prefetch\1102231642.EXE-8311975F.pf
C:\Windows\Prefetch\MDM.EXE-E5C1239F.pf
C:\Windows\Prefetch\WIN.EXE-FE4EAC67.pf
C:\Windows\Prefetch\WIN32.EXE-31D65D18.pf
C:\Windows\Prefetch\WININST.EXE-66A7782D.pf
```

5.6 Process Monitor analysis

After event capture is stopped it is good to save the results for later analyses.

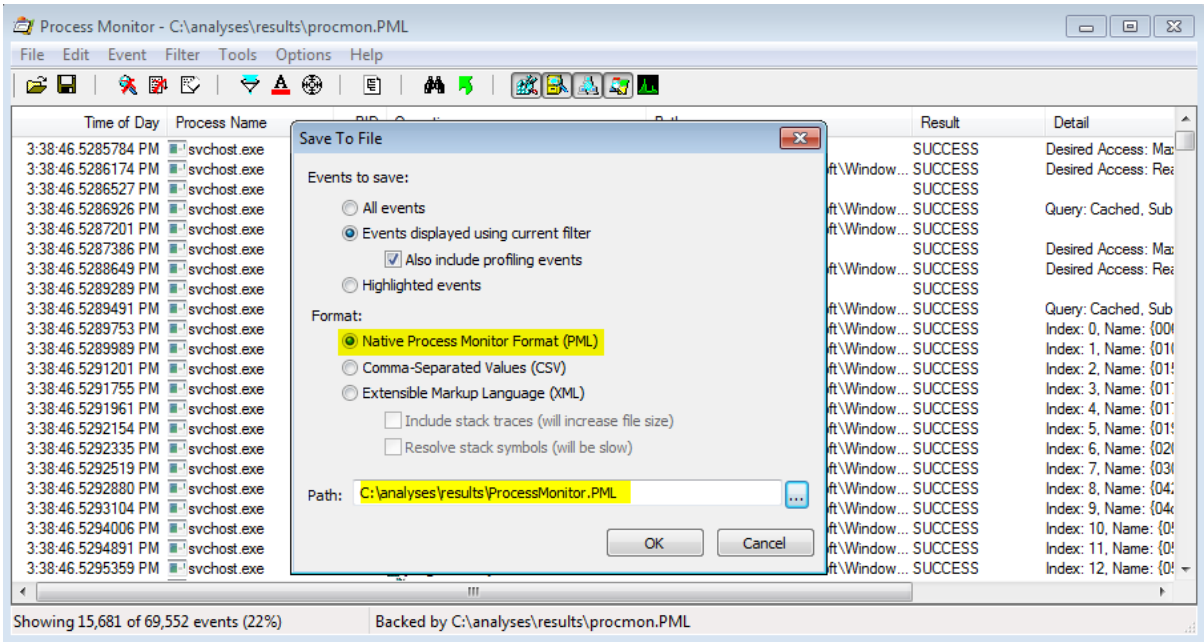


Figure 67. Saving Process Monitor results.

Next using process tree (Tools -> Process Tree...) find suspicious malware processes.

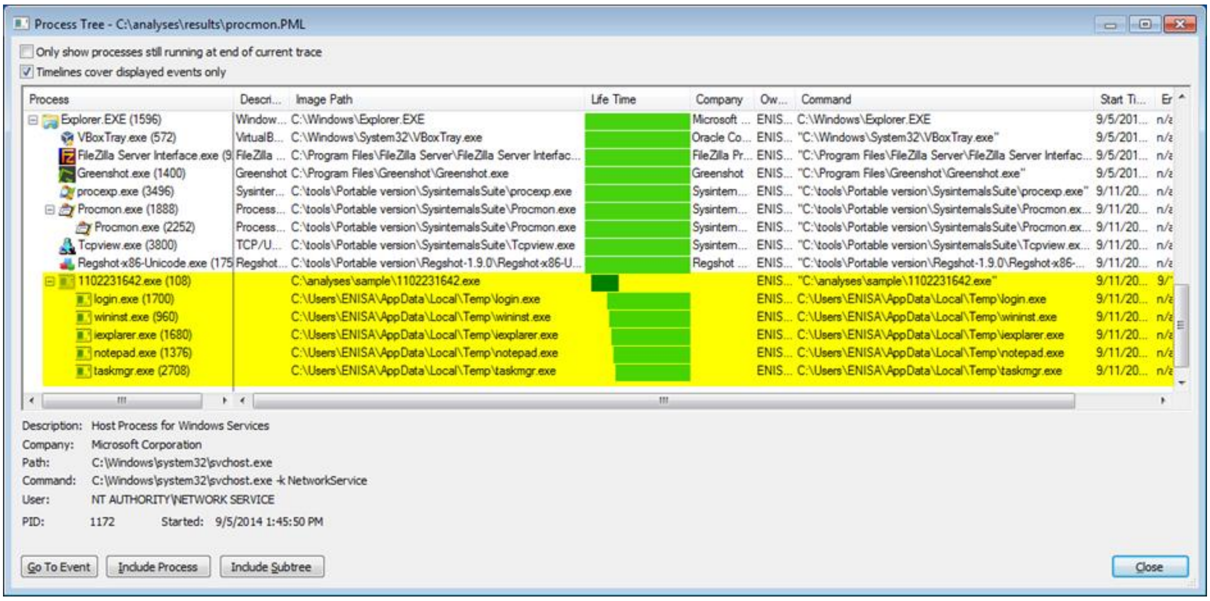


Figure 68. Locating malicious processes using Process Tree.

From analysing the process *Life Time* it is clear that malware process (1102231642.exe) first started, spawned additional child processes and quit. Right click each malware process and choose “Add process to Include filter”. Now only visible events in the main Process Monitor window will be the events related to selected processes.

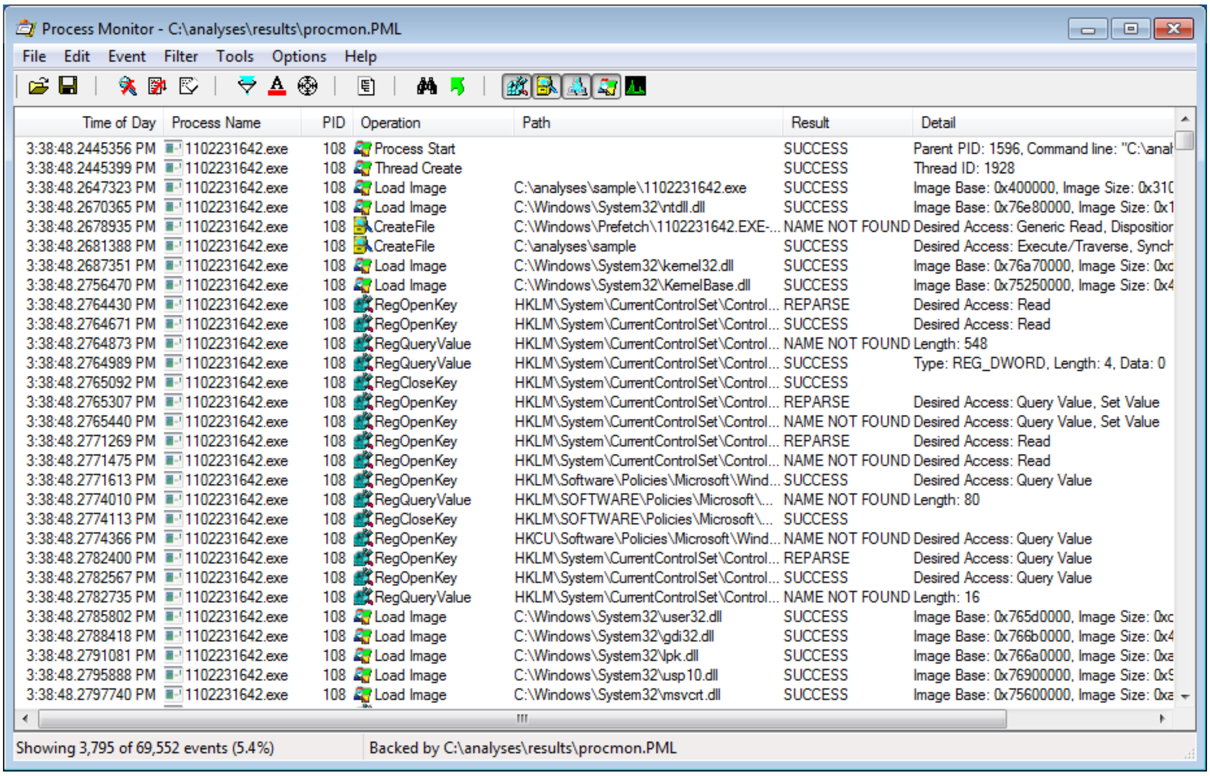


Figure 69. Process Monitor window after filtering out unnecessary processes.

Due to the large amount of information, it is good idea to limit it to only more interesting events. Students can achieve this by either highlighting interesting events or adding them to a filter.

First students should try to highlight the following operations: Process Create, WriteFile, and Process Start. This can be done using Process Monitor Highlighting dialog window (Filter -> Highlight…). An alternate way is to right click on a selected event and choose 'Highlight <name>'.

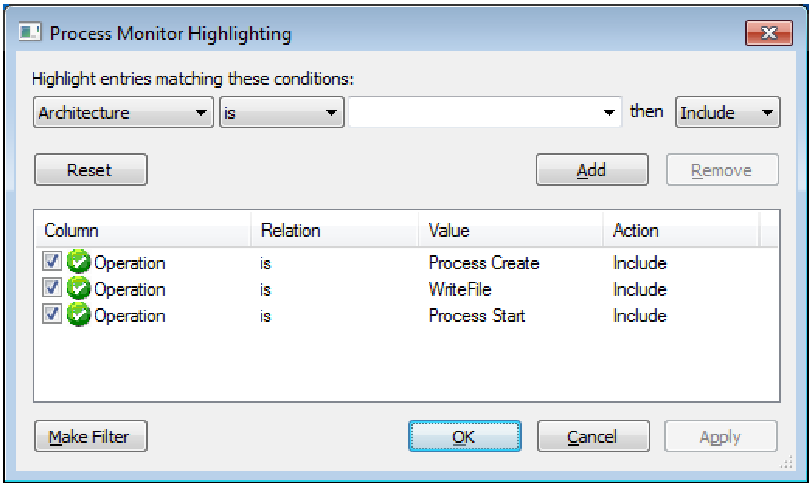


Figure 70. Adding highlight in Process Monitor.

After highlighting filter main Process Monitor window should look similar to the following:

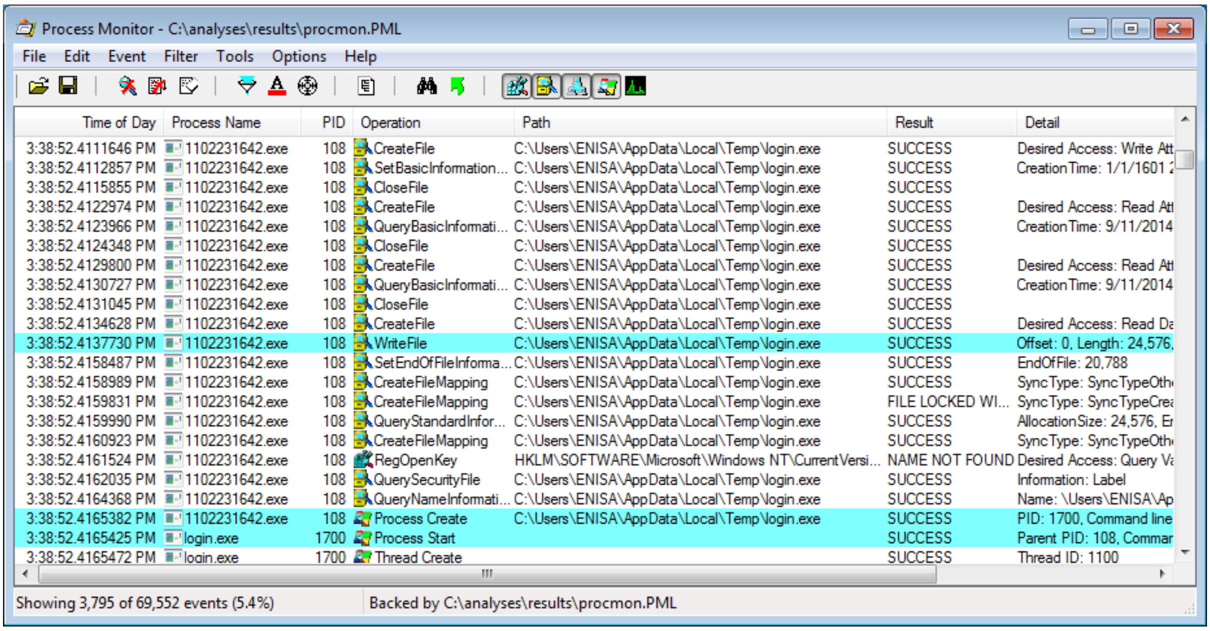


Figure 71. Process Monitor highlight.

Students can now scroll down the events list easily and follow interesting operations.

Next, the students should try to add include filters in the same manner (highlight filter can be now disabled). Operations for include filter: RegSetValue, WriteFile, Process Create. This can be done using Process Monitor Filter dialog (Filter -> Filter…).

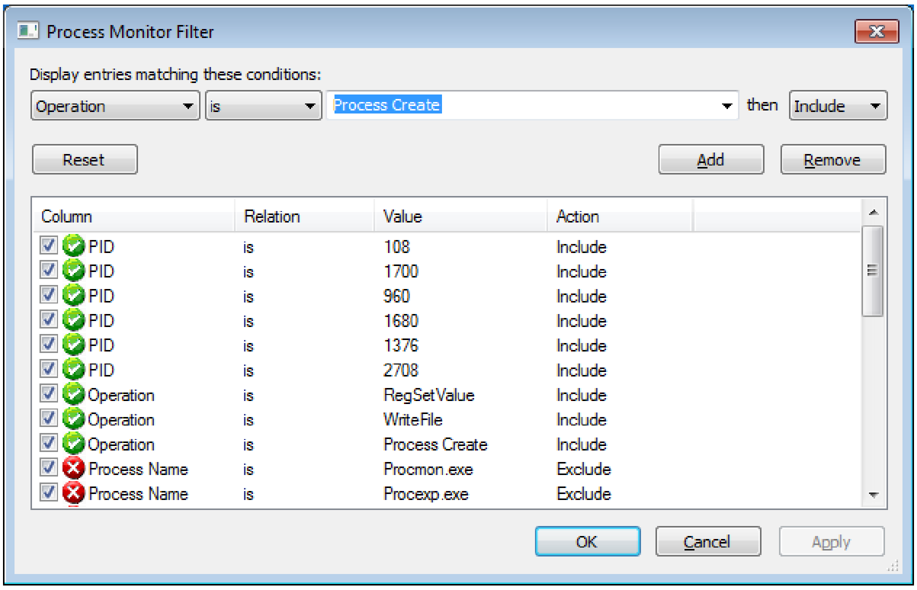


Figure 72. Include filters in Process Monitor.

After applying the include filters, main Process Monitor window should look like:

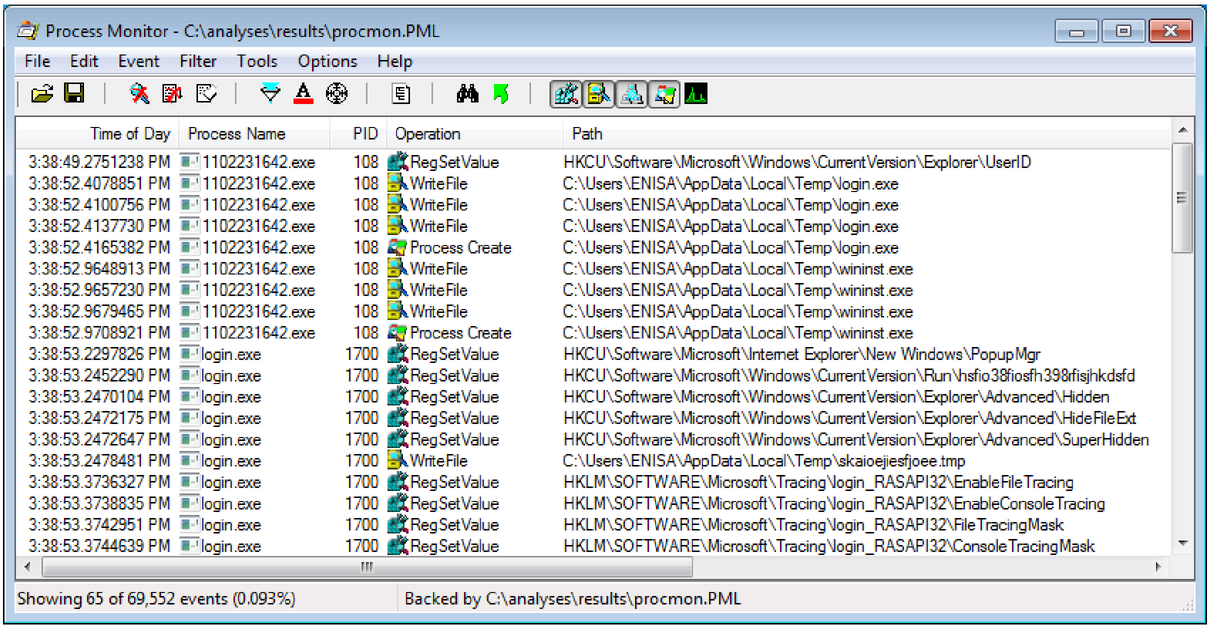


Figure 73. Process Monitor after applying include filter.

Following filtered events, we are able to see that the main malware process isn't responsible for setting persistence and modifying other registry values. It is the first spawned process (in this case login.exe) which installs itself in HKCU\Software\Microsoft\Windows\CurrentVersion\Run\ and also creates .tmp file in %LOCALAPPDATA%.

In general the highlight feature is useful to analyse certain events with respect to other events. For example to check which events progressed with a new process creation, highlight *Process Create* event and then analyse events proceeding each highlighted event. On the other hand, using the include filter is useful when one needs to focus only on a group of events that meet a given criteria and no other events.

Double clicking on each event will reveal additional information. Double click on one of the WriteFile events of the main 1102231642.exe process and switch to the *Stack* tab in the new dialog window.

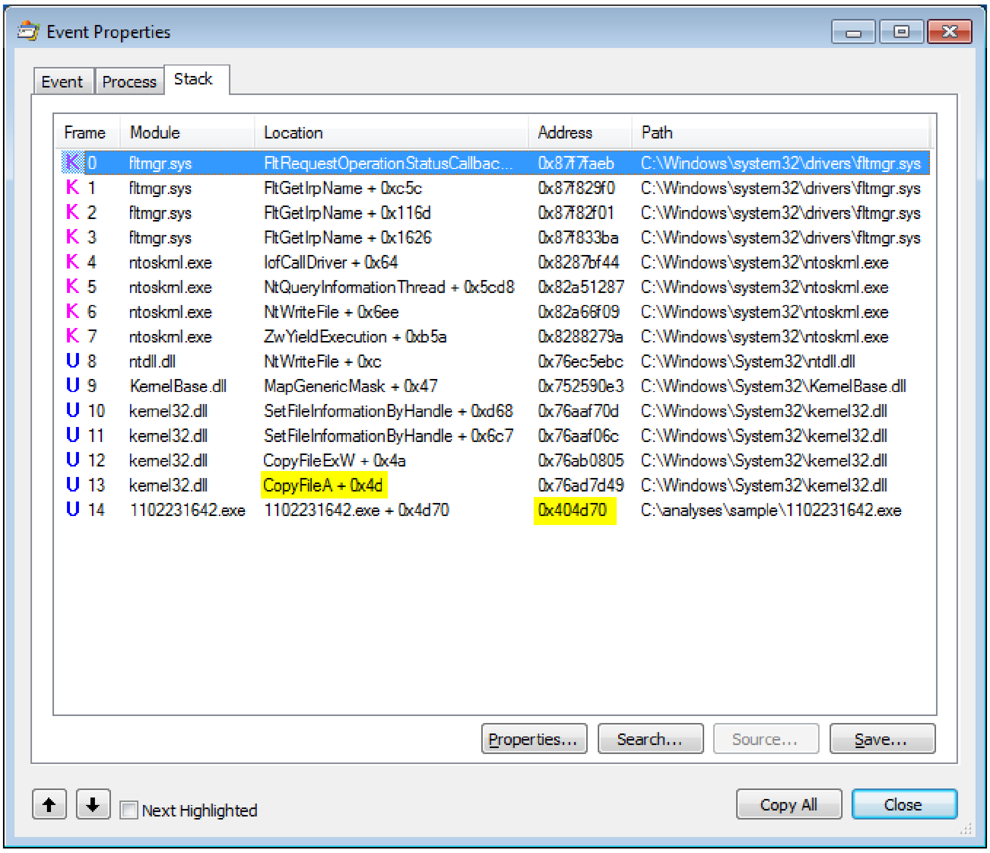


Figure 74. Stack view in Process Monitor

At this window, the student can view the call stack of the calling process at the moment when the event occurred. In this example, the event was a result of the CopyFileA function call from the main malware process. Additional helpful information is the address at which the call took place – 0x404d70. This address can be used during more advanced static analysis to quickly locate the routine responsible for copying new executable files.

Next, the students should view the Cross Reference Summary (Tools -> Cross Reference Summary…). This window shows which files and registry keys were written to or read from, and by what processes.

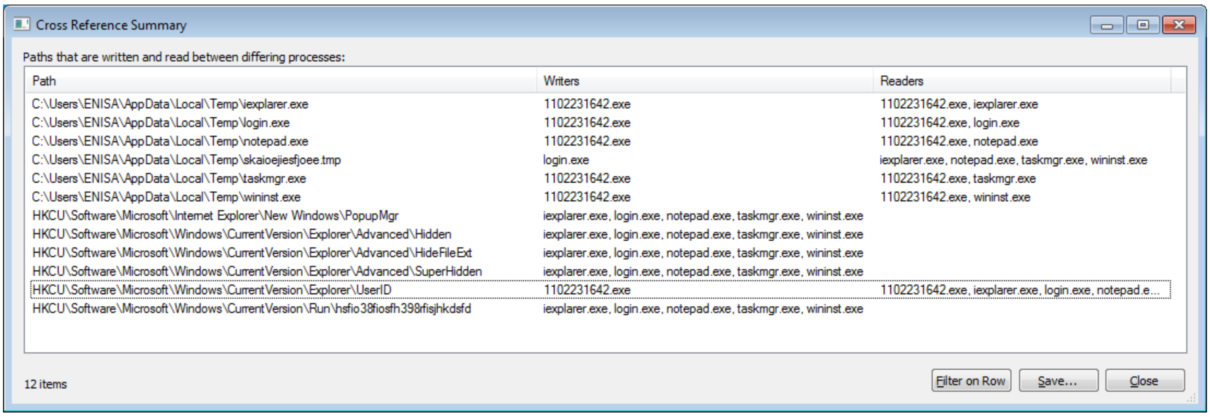


Figure 75. Process Monitor cross reference summary.

We can see that .tmp file is written by only one spawned process. The rest of the processes only read this file. This means that this file might be used for the IPC (Inter Process Communication) of spawned processes. It is also worth to notice the UserID key is written to only by the main malware process, and read by rest of the processes. This means that this key might be used to store configuration data for other processes.

Exercise:

1. **Create filter in Process Monitor which will detect all writes to the .exe files by any system process.**
   To create this filter students need to create two *Include* filters:
   - Operation, is, WriteFile
   - Path, ends with, .exe

Column	Relation	Value	Action
Operation	is	WriteFile	Include
Path	ends with	.exe	Include

Figure 76. Process Monitor filter detecting writes to .exe files.

5.7 Searching for rootkit artifacts

In the final step of the analysis, the students will be searching for rootkit artifacts using GMER tool. Depending on the GMER results, additional analysis steps may be taken – for example if GMER detects new hidden file that wasn't detected in any of the previous steps.

First close all open tools used in the first part of the exercise (Process Explorer, Process Monitor, etc.) and then start GMER.

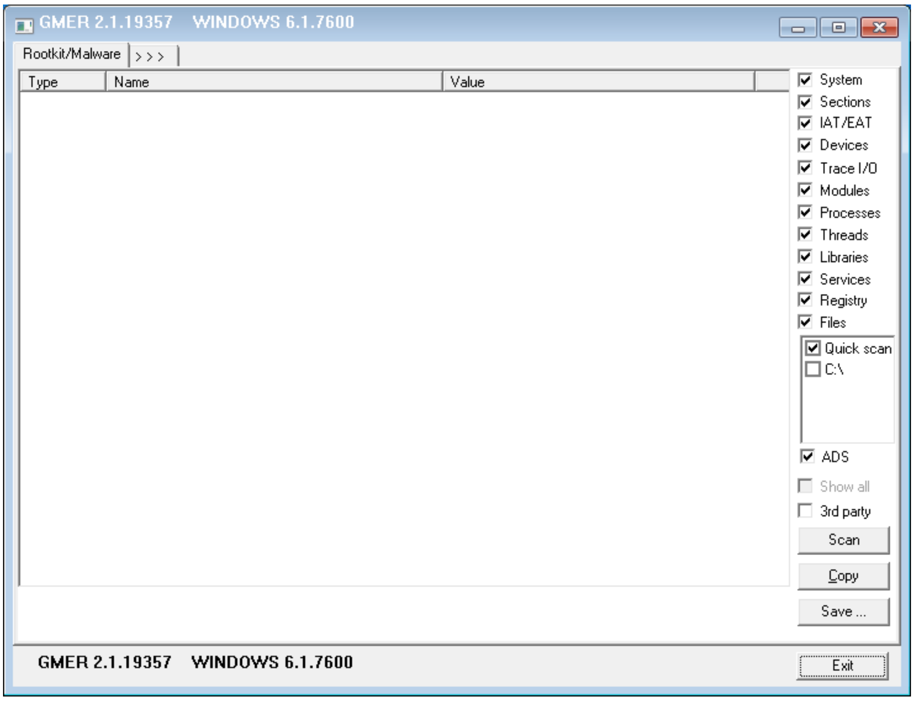


Figure 77. Main GMER window.

Leaving the default analysis options set (*System, Sections, IAT/EAT, etc.*) click *Scan* to begin system scanning. Depending on the VM size and resources, analysis might take some time (up to several minutes). Sometimes, to speed up the scanning, a user might decide to choose fewer analysis options.

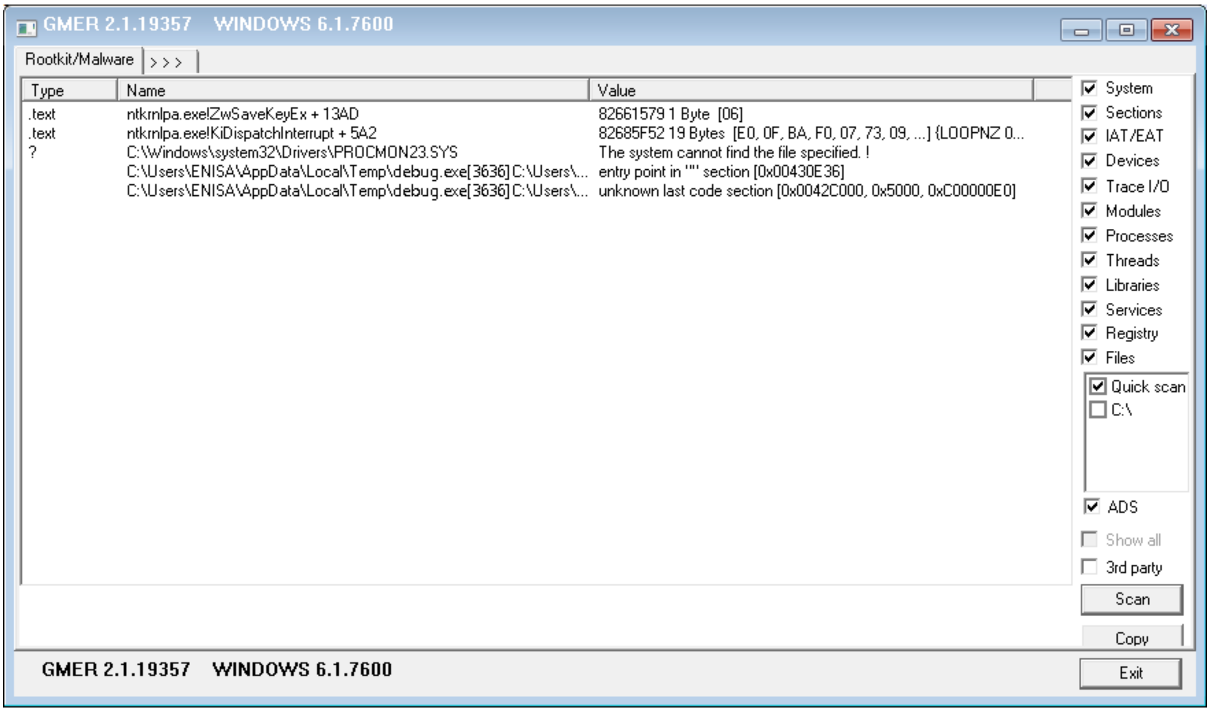


Figure 78. GMER results.

In this case, the first three changes reported by GMER (two hooks and a file system problem) are changes that are always reported by GMER on this system. An additional two changes report a suspicious structure of the debug.exe which indicate that some obfuscation was used. There are no changes indicating typical rootkit activity (e.g. hooks on many system functions, hidden files, and hidden processes). Note that running GMER more than once can produce additional hits, for instance files in a temporary directory that can be created during previous runs by the tool itself.

5.8 Finishing analysis

After the analysis is finished, copy all of the results obtained, screenshots, and notes to the directory: C:\analyses\results, and send them to Styx as described in the task: *Basic static analysis*.

After the results are sent to Styx, shutdown Winbox machine and restore the clean snapshot.

5.9 Extra samples

As an extra exercise, students can analyse additional malware samples using techniques in this task. Extra samples names are: dddsf.exe, inst2.exe, msupdate.exe. Samples can be found in /home/enisa/enisa/ex3/extra.

It is not necessary to stick precisely to the behavioural analysis algorithm described in this task. Students might use only some of the tools described or use tools not described in this task, but present on the Winbox machine (if they are familiar with them, e.g. Rohitab API Monitor, OllyDbg). Students are advised to use snapshots during the analysis. For each sample it should be possible to point to some of its functionality. After each analysis, students should have an open discussion to share their findings.

6 Task 3: Network analysis

In this task students will capture and analyse network traffic generated by malware. The first step shows how to conduct network analysis, and obtain three types of network analysis results: network traffic capture in PCAP format, MITMProxy capture log, and INetSim log files. The next steps will cover various types of network traffic generated by three different malware samples.

The network type used in all following analyses will be *netsim_mitmproxy*. Students should also remember that not all traffic captured in the exercise is explicitly generated by malware. Depending on the Windows version and configuration on the Winbox machine there might be some traffic captured that is not related to the malware.

6.1 Network traffic capture and log acquisition

First, restore the Winbox snapshot used for dynamic analyses and send the malware sample to the Winbox. In this step, use sample *pz_7.exe* which will be also used in the next step. If the sample is not already present in Viper it can be found in the directory: /home/enisa/enisa/ex3/samples.

```
enisa viper > find name pz_7
+---+----------+----------------------+----------------------------------+-------+
| # | Name     | Mime                 | MD5                              | Tags  |
+---+----------+----------------------+----------------------------------+-------+
| 1 | pz_7.exe | application/x-dosexec | cd77c349ad031dbc9bba76daa72a24cb | shiva |
+---+----------+----------------------+----------------------------------+-------+
enisa viper > open -l 1
[*] Session opened on /opt/viper/projects/enisa/binaries/4/c/d/3/4cd3bc38daf20e26872784bf75478041a0
4d1d124adffb6c7973723150078afb
enisa viper pz_7.exe > lab-send
226 Successfully transferred "/sample/pz_7.exe"
enisa viper pz_7.exe >
```

Figure 79. Sending sample to the analysis.

After restoring the virtual machine and sending the sample make sure that *netsim_mitmproxy* network type is currently chosen.

Switching network configuration to netsim_mitmproxy

```
$ lab-switch-net netsim_mitmproxy
Applying changes...
```

Next clean all network related logs and result files (Inetsim, MITMProxy, Snort, pcaps) using lab-cleanlogs script. It is necessary because there might be some logs left from a previous analyses.

Cleaning old logs

```
$ lab-cleanlogs
```

When the logs are deleted, start the network traffic capture (PCAP) and MITMProxy tool. Pcap files and MITMProxy logs will be automatically saved to separate files in /lab/var directory.

Starting network capture and mitmproxy

```
$ lab-netdump start
Starting capture to /lab/var/pcaps/net_140922115236.pcap
$ lab-mitmproxy
```

Figure 80. New MITMProxy window (with no logs captured yet)

After starting the network capture switch to the Winbox window, execute the malware sample and wait for a few minutes. It is good to let the malware run for at least 4-5 minutes, but the ideal time might differ according to the malware sample or malware family. In general the goal is to capture all different types of network traffic generated by the malware. Usually at some point in time, the network actions performed by the malware starts repeating periodically or stops. This will be the indicator that there is no need to capture more network traffic. One should also be able to recognize network patterns resulting from some dynamic or random generator. Example of such traffic might be DGA (Domain Generation Algorithm) when malware tries to connect to dynamically generated domain names. In such situation capturing a limited number of such domains will be enough.

During the exercise it is not necessary to wait until network traffic starts repeating. Waiting about 4-5 minutes should be enough for all samples.

Optionally, to view live capture of the network traffic, students might decide to open a new Styx console window (either connecting to Styx via SSH or using screen to start MITMProxy) and then start reading .pcap file with Tcpdump (pcap filename should be replaced with the actual one).

Viewing live network capture

```
$ cd /lab/var
$ tail -c 100000000 -f net_140922115236.pcap | tcpdump -nr-
```

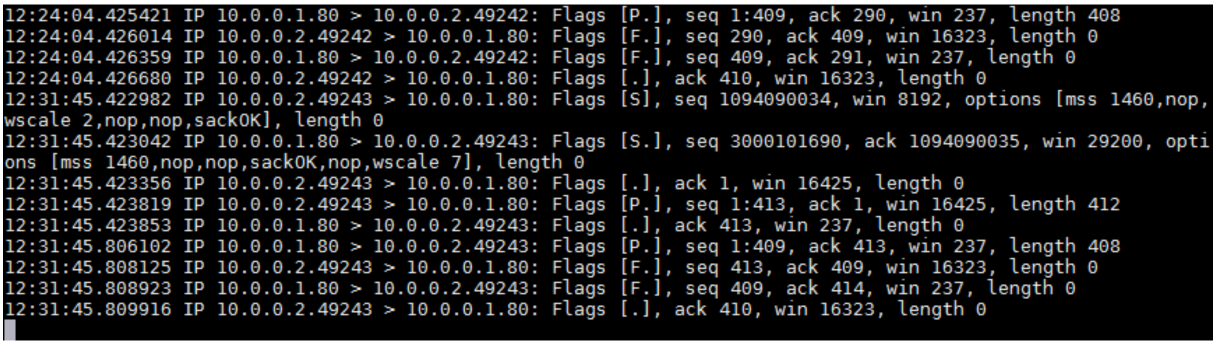


Figure 81. Live view of the network traffic capture.

Students might also decide to run Wireshark inside the Winbox machine. In most situations it will work without any problem, but in rare cases, sophisticated malware might try to evade network capture inside the Winbox machine, or detect Wireshark and change its behaviour. For samples used in this exercise, students should not have a problem using Wireshark inside Winbox.

After enough time elapses (4-5 minutes), stop mitmproxy capture by pressing ‘q’ (*quit*) key and then ‘y’ (*yes*). MITMProxy will save results to /lab/var/mitmproxy/mitm.dump.

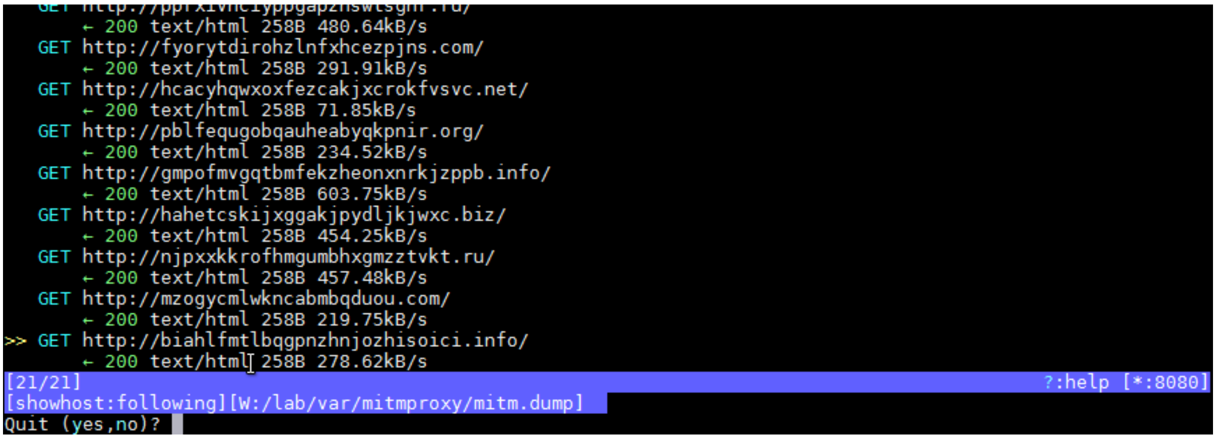


Figure 82. Quiting mitmproxy.

Then stop *tcpdump* packet capture and restart INetSim service. Restarting INetSim service is necessary for INetSim to generate report summarizing observed traffic.

Stopping *tcpdump* packet capture

```
$ lab-netdump stop
Capture stopped [/lab/var/pcaps/net_140922115733.pcap]
$ sudo service inetsim restart
* Restarting Internet Service Simulation Suite inetsim
   ...done.
```

Then copy all result files for further analysis:

Stopping *tcpdump* packet capture

```
$ mkdir -p /lab/analyses/pz_7.exe
$ cd /lab/analyses/pz_7.exe
$ sudo cp -a ../../var net_results
```

Now, the network traffic capture and log acquisition is finished and the students can restore the clean snapshot of the Winbox machine.

6.2 P2P and DGA traffic

In this step sample pz_7z.exe will be analysed. Use network traffic capture obtained in the previous step or send the sample to the Winbox machine and perform a new analysis as described in the previous step. It is also assumed that the result files are stored in the /lab/analyses/pz_7.exe/net_results/ directory.

In case there were any problems with performing analysis, result files can be also obtained from: /home/enisa/enisa/ex3/results/net1/net_results/ directory.

First, start the clean Winbox machine and send to it the pcap file obtained from malware analysis.

```
enisa@styx:/lab$ cd analyses/pz_7.exe/net_results/pcaps/
enisa@styx:/lab/analyses/pz_7.exe/net_results/pcaps$ lab-sendfile net_140919164301.pcap
net_140919164301.pcap:                    153.51 kB   323.17 kB/s
enisa@styx:/lab/analyses/pz_7.exe/net_results/pcaps$
```

Figure 83. Sending pcap to Winbox machine.

Open the uploaded file in Wireshark on Winbox.

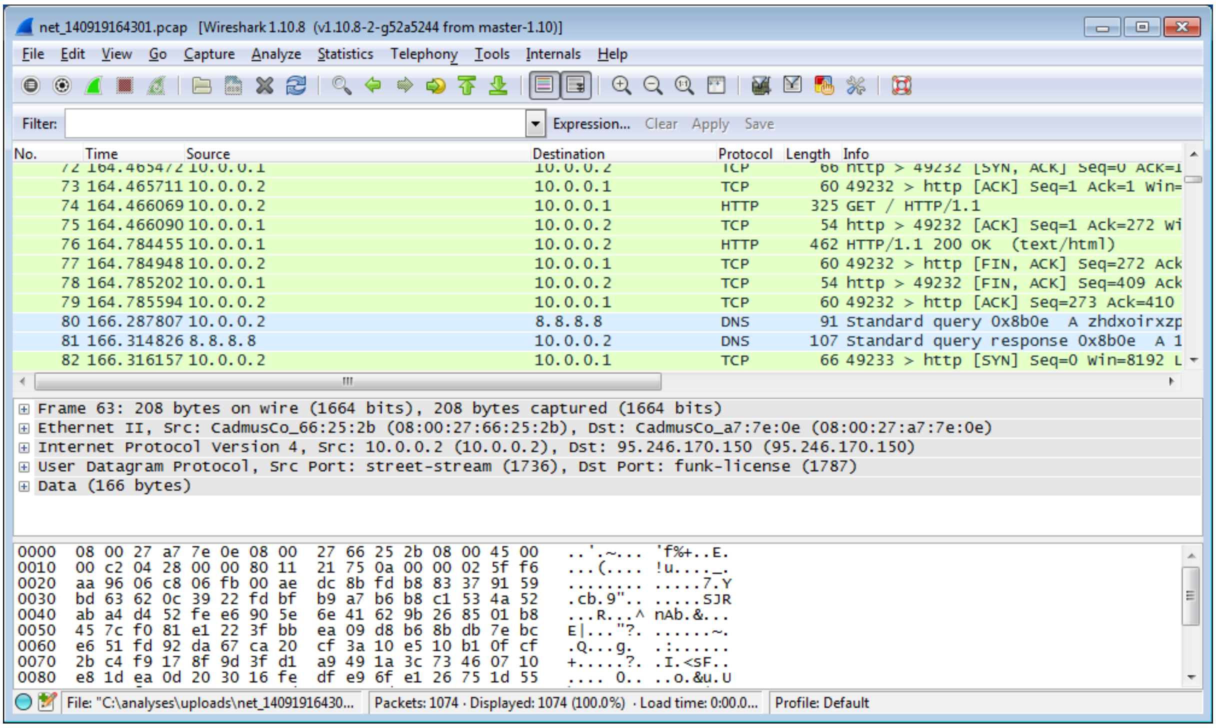
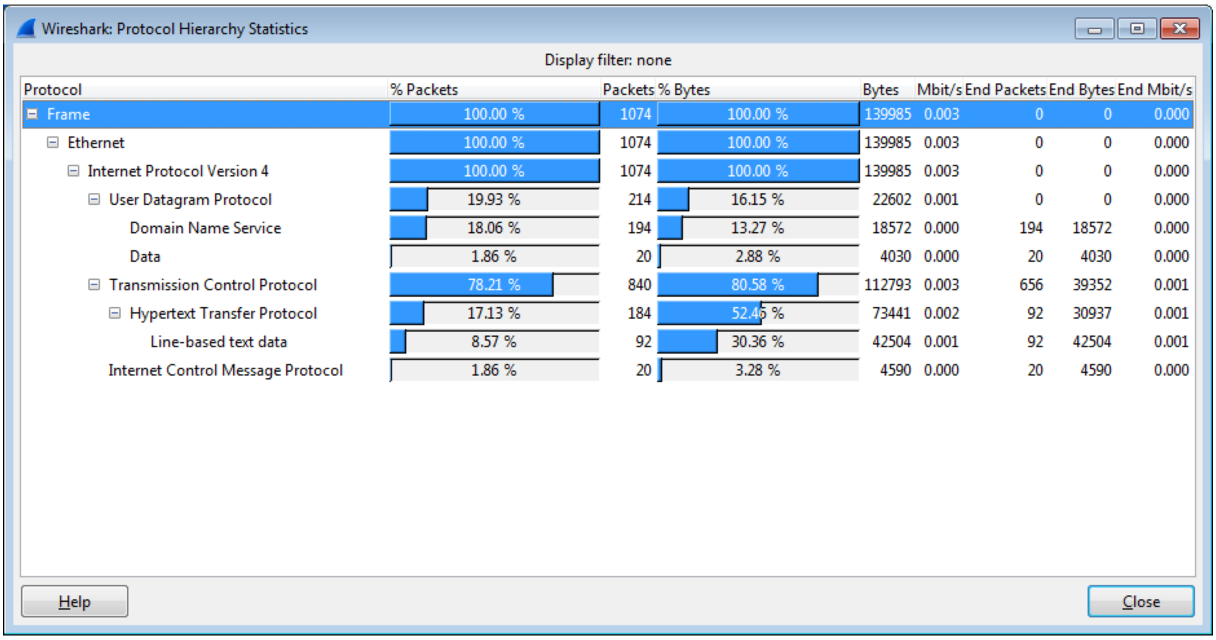


Figure 84. Wireshark window after opening .pcap file.

If there is a lot of captured traffic, it is good to check *Protocol Hierarchy Statistics* to determine what protocols are present in the capture. Otherwise it is sometimes easy to miss protocols for which only a few packets were sent.

To view *Protocol Hierarchy Statistics* choose *Protocol Hierarchy* from the *Statistics* menu.


Wireshark: Protocol Hierarchy Statistics

Display filter: none

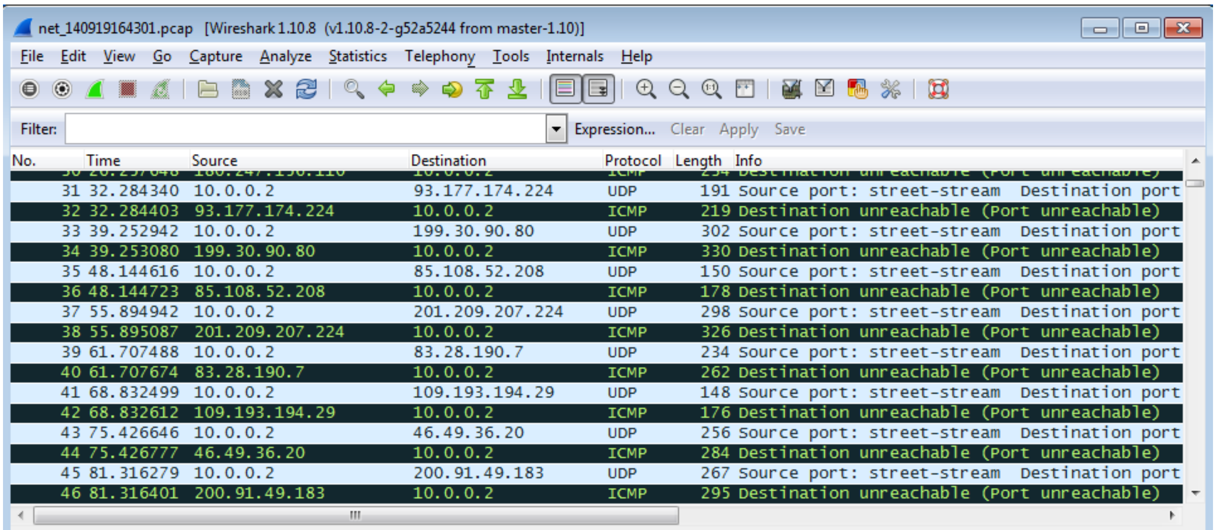
Protocol	% Packets	Packets	% Bytes	Bytes	Mbit/s	End Packets	End Bytes	End Mbit/s
Frame	100.00 %	1074	100.00 %	139985	0.003	0	0	0.000
Ethernet	100.00 %	1074	100.00 %	139985	0.003	0	0	0.000
Internet Protocol Version 4	100.00 %	1074	100.00 %	139985	0.003	0	0	0.000
User Datagram Protocol	19.93 %	214	16.15 %	22602	0.001	0	0	0.000
Domain Name Service	18.06 %	194	13.27 %	18572	0.000	194	18572	0.000
Data	1.86 %	20	2.88 %	4030	0.000	20	4030	0.000
Transmission Control Protocol	78.21 %	840	80.58 %	112793	0.003	656	39352	0.001
Hypertext Transfer Protocol	17.13 %	184	52.45 %	73441	0.002	92	30937	0.001
Line-based text data	8.57 %	92	30.36 %	42504	0.001	92	42504	0.001
Internet Control Message Protocol	1.86 %	20	3.28 %	4590	0.000	20	4590	0.000

Help Close

Figure 85. Viewing protocol hierarchy statistics.

As we can see communication mostly consisted of HTTP traffic, DNS requests, some unknown UDP datagrams (UDP data) and also some ICMP messages.

Next close *Protocol Hierarchy Statistics* and go back to main Wireshark window. Scroll down till you see some UDP traffic.


net_140919164301.pcap [Wireshark 1.10.8 (v1.10.8-2-g52a5244 from master-1.10)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
31	32.284340	10.0.0.2	93.177.174.224	UDP	191	Source port: street-stream Destination port
32	32.284403	93.177.174.224	10.0.0.2	ICMP	219	Destination unreachable (Port unreachable)
33	39.252942	10.0.0.2	199.30.90.80	UDP	302	Source port: street-stream Destination port
34	39.253080	199.30.90.80	10.0.0.2	ICMP	330	Destination unreachable (Port unreachable)
35	48.144616	10.0.0.2	85.108.52.208	UDP	150	Source port: street-stream Destination port
36	48.144723	85.108.52.208	10.0.0.2	ICMP	178	Destination unreachable (Port unreachable)
37	55.894942	10.0.0.2	201.209.207.224	UDP	298	Source port: street-stream Destination port
38	55.895087	201.209.207.224	10.0.0.2	ICMP	326	Destination unreachable (Port unreachable)
39	61.707488	10.0.0.2	83.28.190.7	UDP	234	Source port: street-stream Destination port
40	61.707674	83.28.190.7	10.0.0.2	ICMP	262	Destination unreachable (Port unreachable)
41	68.832499	10.0.0.2	109.193.194.29	UDP	148	Source port: street-stream Destination port
42	68.832612	109.193.194.29	10.0.0.2	ICMP	176	Destination unreachable (Port unreachable)
43	75.426646	10.0.0.2	46.49.36.20	UDP	256	Source port: street-stream Destination port
44	75.426777	46.49.36.20	10.0.0.2	ICMP	284	Destination unreachable (Port unreachable)
45	81.316279	10.0.0.2	200.91.49.183	UDP	267	Source port: street-stream Destination port
46	81.316401	200.91.49.183	10.0.0.2	ICMP	295	Destination unreachable (Port unreachable)

Figure 86. UDP traffic in Wireshark window.

This is clearly not normal traffic generated by the operating system. Such traffic is usually characteristic to malware with P2P functionality using protocols like Kademlia. Also the fact that the malware is trying to connect to the external IP addresses means that those addresses were either hardcoded or dynamically generated by the malware (because any DNS requests resolve to 10.0.0.1 in this laboratory).

To further inspect udp traffic, apply the following Wireshark view filter: *ip.src == 10.0.0.2 && udp && !icmp*

Filter: ip.src == 10.0.0.2 && udp && !icmp   Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
18	22.826025	10.0.0.2	8.8.8.8	DNS	77	Standard query 0x74ce A crl.microsoft.com
29	26.257530	10.0.0.2	180.247.156.110	UDP	206	Source port: street-stream Destination port: 24609
31	32.284340	10.0.0.2	93.177.174.224	UDP	191	Source port: street-stream Destination port: xrpc-registry
33	39.252942	10.0.0.2	199.30.90.80	UDP	302	Source port: street-stream Destination port: 7761
35	48.144616	10.0.0.2	85.108.52.208	UDP	150	Source port: street-stream Destination port: 4627
37	55.894942	10.0.0.2	201.209.207.224	UDP	298	Source port: street-stream Destination port: qmvideo
39	61.707488	10.0.0.2	83.28.190.7	UDP	234	Source port: street-stream Destination port: 12498
41	68.832499	10.0.0.2	109.193.194.29	UDP	148	Source port: street-stream Destination port: 7057
43	75.426646	10.0.0.2	46.49.36.20	UDP	256	Source port: street-stream Destination port: 9752
45	81.316279	10.0.0.2	200.91.49.183	UDP	267	Source port: street-stream Destination port: 7399
47	86.377774	10.0.0.2	84.59.131.0	UDP	144	Source port: street-stream Destination port: 7605
49	92.159005	10.0.0.2	94.240.216.82	UDP	181	Source port: street-stream Destination port: queueadm
51	98.129072	10.0.0.2	108.74.172.39	UDP	164	Source port: street-stream Destination port: qsoft
53	105.535522	10.0.0.2	107.193.222.108	UDP	162	Source port: street-stream Destination port: starfish
55	114.191200	10.0.0.2	94.240.224.115	UDP	155	Source port: street-stream Destination port: 8696
57	121.349339	10.0.0.2	107.217.117.139	UDP	246	Source port: street-stream Destination port: 8593
59	127.551482	10.0.0.2	76.226.114.217	UDP	119	Source port: street-stream Destination port: snaresecure
61	134.176071	10.0.0.2	201.209.58.176	UDP	162	Source port: street-stream Destination port: 14191
63	142.082792	10.0.0.2	95.246.170.150	UDP	208	Source port: street-stream Destination port: funk-license
65	150.253787	10.0.0.2	123.238.67.140	UDP	269	Source port: street-stream Destination port: 4636
67	157.348625	10.0.0.2	5.20.67.209	UDP	168	Source port: street-stream Destination port: ttg-protocol
69	164.431849	10.0.0.2	8.8.8.8	DNS	74	Standard query 0x0ded A www.google.com

Figure 87. Wireshark after applying view filter.

Then compare UDP packets with each other – checking source and destination ports, UDP payload size and content.

```
⊞ Frame 29: 206 bytes on wire (1648 bits), 206 bytes captured (1648 bits)
⊞ Ethernet II, Src: CadmusCo_66:25:2b (08:00:27:66:25:2b), Dst: CadmusCo_a7:7e:0e (08:00:27:a7:7e:0e)
⊞ Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 180.247.156.110 (180.247.156.110)
⊞ User Datagram Protocol, Src Port: street-stream (1736), Dst Port: 24609 (24609)
⊞ Data (164 bytes)
```

Figure 88. UDP datagram sent to 180.247.156.110.

```
⊞ Frame 31: 191 bytes on wire (1528 bits), 191 bytes captured (1528 bits)
⊞ Ethernet II, Src: CadmusCo_66:25:2b (08:00:27:66:25:2b), Dst: CadmusCo_a7:7e:0e (08:00:27:a7:7e:0e)
⊞ Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 93.177.174.224 (93.177.174.224)
⊞ User Datagram Protocol, Src Port: street-stream (1736), Dst Port: xrpc-registry (3651)
⊞ Data (149 bytes)
```

Figure 89. UDP datagram to 93.177.174.224.

We can observe that each UDP datagram is addressed to a different destination port but originates from the same source port number - 1736. Also, the payload size seems to be different for each datagram.

Analysing datagrams payloads we see that there are no common bytes and all the content seems to be randomized. This means that the malware is likely using some sort of encryption resulting in different content for each datagram. Differences in size of payloads suggest that malware might be also adding some padding or junk bytes in the protocol.

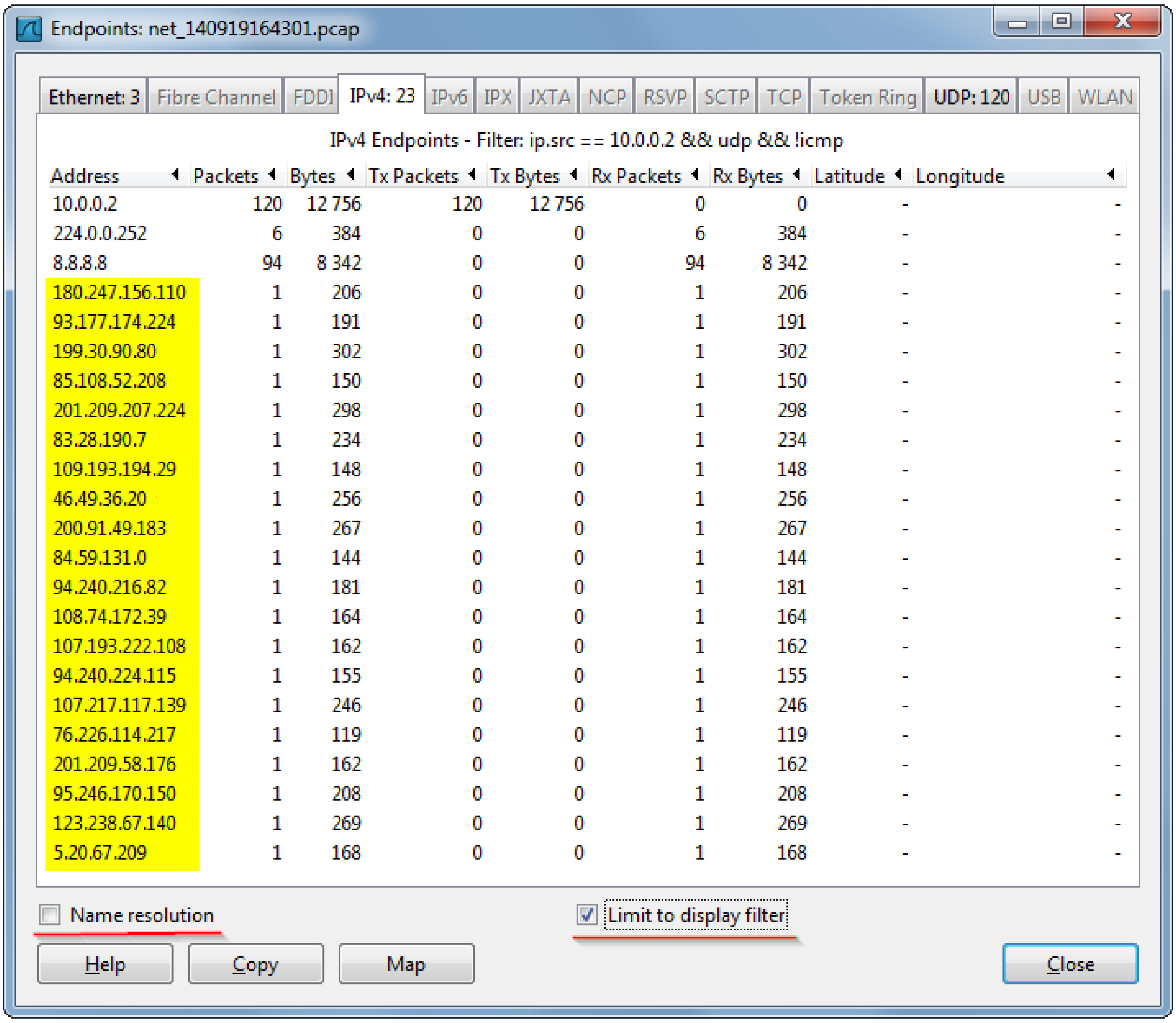
```
0000  08 00 27 a7 7e 0e 08 00  27 66 25 2b 08 00 45 00   ..'.~... 'f%+..E.
0010  00 c0 04 17 00 00 80 11  da ae 0a 00 00 02 b4 f7   ........ ........
0020  9c 6e 06 c8 60 21 00 ac  34 a2 aa cc 9f bf 39 b3   .n..`!.. 4.....9.
0030  24 42 a2 6f 4d 22 3a 82  68 2d 34 3f 9c 32 58 3e   $B.oM":. h-4?.2X>
0040  11 b9 a7 64 60 9e d1 ea  6b 20 75 56 b6 e0 82 54   ...d`... k uV...T
0050  9d 4a 40 c8 da ac b8 fd  ca 6d ab cb fa 5e 2f ce   .J@..... .m...^/.
0060  82 1b 95 a8 42 73 a7 30  a7 e4 29 c7 70 5b 4a 97   ....Bs.0 ..).p[J.
0070  01 22 09 83 ef af 86 95  c1 66 2b 11 41 52 80 f9   ."...... .f+.AR..
0080  55 66 03 e3 c7 18 ae 46  73 61 22 44 14 84 2d 6a   Uf.....F sa"D..-j
0090  39 07 a5 ca 71 1f 5d 80  b7 ed 63 65 55 3a 76 01   9...q.]. ..ceU:v.
00a0  88 fe 51 85 37 9a 41 62  cb c7 de 3b 4f f0 0f 07   ..Q.7.Ab ...;O...
00b0  91 13 50 cf 57 58 94 6f  4b f2 d5 29 e4 2b 62 73   ..P.WX.o K..).+bs
00c0  46 6f dc 13 d7 9d 3a a9  46 61 b0 82 d7 36         Fo....:. Fa...6
```

Figure 90. Payload of datagram addressed to 180.247.156.110.

```
0000  08 00 27 a7 7e 0e 08 00  27 66 25 2b 08 00 45 00   ..'.~... 'f%+..E.
0010  00 b1 04 18 00 00 80 11  1f 91 0a 00 00 02 5d b1   ........ ......].
0020  ae e0 06 c8 0e 43 00 9d  81 49 5e e5 97 37 e7 42   .....C.. .I^..7.B
0030  eb 09 b9 b4 7b 2b bb c9  fc 08 21 05 ed 65 10 38   ....{+.. ..!..e.8
0040  b7 63 46 53 b9 db 57 77  a7 f8 66 00 f5 a8 28 e8   .cFS..Ww ..f...(.
0050  75 eb cb 6a 1c b5 a8 77  39 4d 04 8c c1 e8 98 a8   u..j...w 9M......
0060  61 8f 2d ce 66 7d 47 f7  cd 41 67 09 b4 5f 94 47   a.-.f}G. .Ag.._.G
0070  6b 5c 33 60 c2 c9 13 d8  d6 b4 30 97 50 d2 8c 53   k\3`.... ..0.P..S
0080  20 2d b3 0f 6e 5b 00 a2  6e ad 62 3f 79 4b 43 a4    -..n[.. n.b?yKC.
0090  85 48 92 8c c1 a1 15 1f  b6 ac a5 fd 70 a6 45 a9   .H...... ....p.E.
00a0  35 a5 5b 4d 25 4b 83 29  15 71 4a 60 4e c4 6c 06   5.[M%K.) .qJ`N.l.
00b0  49 45 d3 a9 fa b8 12 3d  a4 32 61 2a bf 08 7b      IE.....= .2a*..{
```

Figure 91. Payload of datagram addressed to 93.177.174.224.

To get a distinct list of IP addresses to which malware sent datagrams, select *Endpoints* module from *Statistics* menu (without clearing Wireshark filter). Then switch to IP tab, check *Limit to display filter* and uncheck *Name resolution.*



Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Latitude	Longitude
10.0.0.2	120	12 756	120	12 756	0	0	-	-
224.0.0.252	6	384	0	0	6	384	-	-
8.8.8.8	94	8 342	0	0	94	8 342	-	-
180.247.156.110	1	206	0	0	1	206	-	-
93.177.174.224	1	191	0	0	1	191	-	-
199.30.90.80	1	302	0	0	1	302	-	-
85.108.52.208	1	150	0	0	1	150	-	-
201.209.207.224	1	298	0	0	1	298	-	-
83.28.190.7	1	234	0	0	1	234	-	-
109.193.194.29	1	148	0	0	1	148	-	-
46.49.36.20	1	256	0	0	1	256	-	-
200.91.49.183	1	267	0	0	1	267	-	-
84.59.131.0	1	144	0	0	1	144	-	-
94.240.216.82	1	181	0	0	1	181	-	-
108.74.172.39	1	164	0	0	1	164	-	-
107.193.222.108	1	162	0	0	1	162	-	-
94.240.224.115	1	155	0	0	1	155	-	-
107.217.117.139	1	246	0	0	1	246	-	-
76.226.114.217	1	119	0	0	1	119	-	-
201.209.58.176	1	162	0	0	1	162	-	-
95.246.170.150	1	208	0	0	1	208	-	-
123.238.67.140	1	269	0	0	1	269	-	-
5.20.67.209	1	168	0	0	1	168	-	-

Figure 92. UDP endpoints list.

On the above screenshot list of UDP endpoints was marked with yellow colour. We can see that there was only one datagram sent to each endpoint. As of rest IP addresses 10.0.0.2 is local address, 224.0.0.252 is standard multicast address and 8.8.8.8 is primary DNS address.

Next, close the Endpoints window and clear the Wireshark filter to get a list of all captured traffic. Then scroll down below to UDP communication. There should be some DNS requests and HTTP communication.

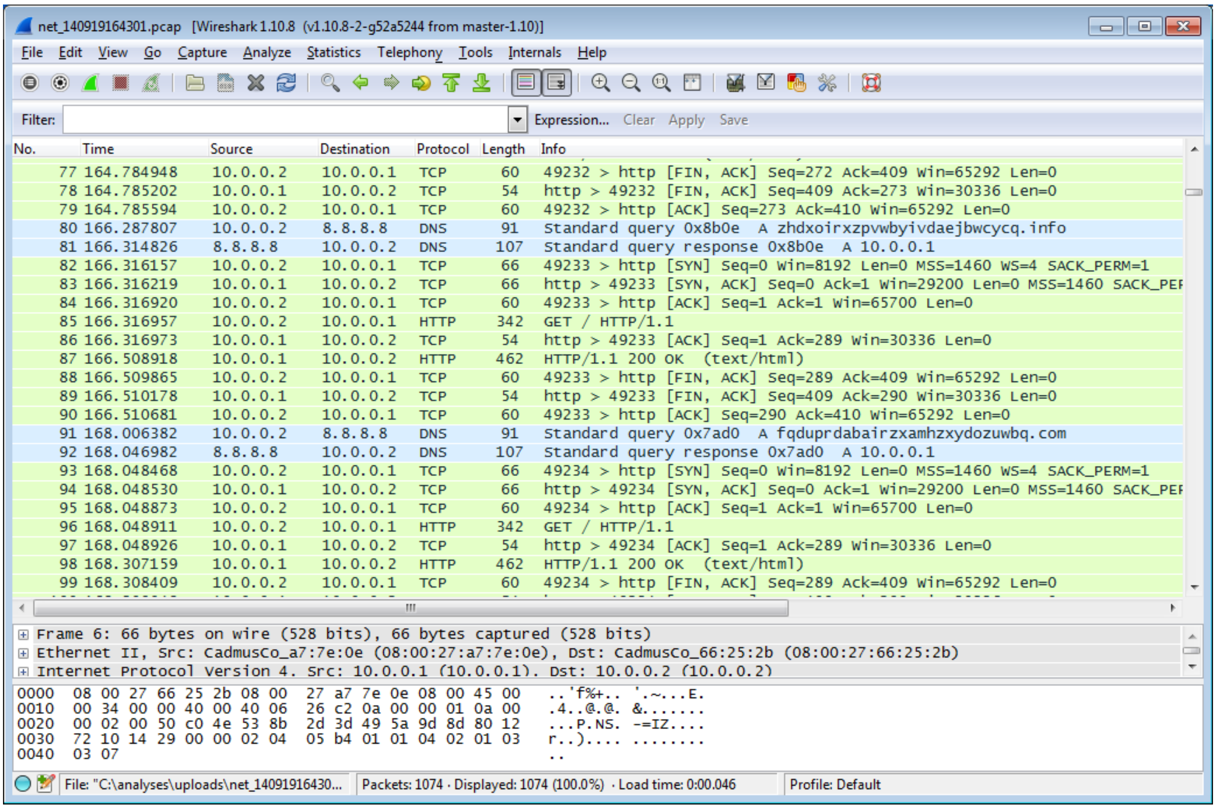


Figure 93. Malware DNS and HTTP communication.

Here we see that the malware is doing DNS requests for random-looking domain names and then connecting to them with HTTP protocol doing GET / request.

To better inspect requested domain names apply the following Wireshark filter: ip.src == 10.0.0.2 && dns

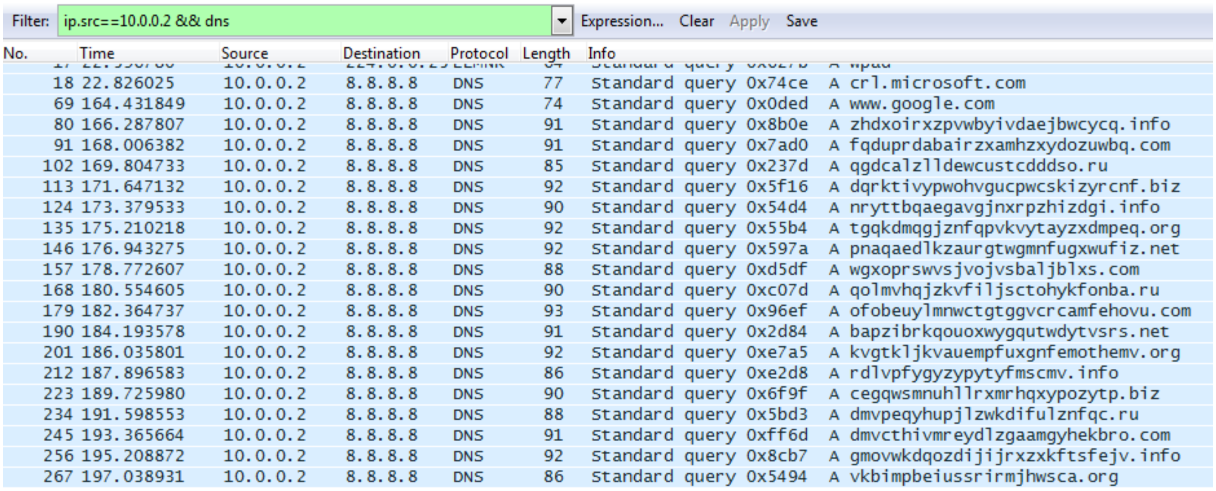
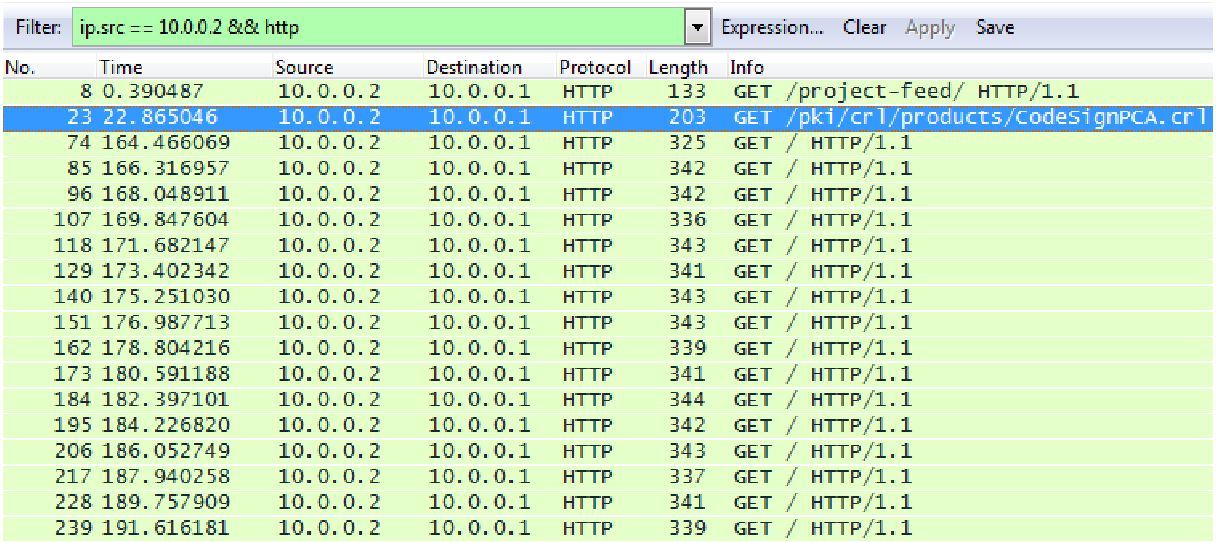


Figure 94. DNS requests filtered in Wireshark window.

This is typical DGA (*Domain Generation* Algorithm) mechanism in which malware is generating seemingly random domain names with some deterministic algorithm and then trying to connect to them. Thanks to DGA, the malware is not limited to hardcoded domain names which can be easily blocked by law enforcement authorities. On the other hand not all DGA domains are registered by botmaster. This means that knowing DGA algorithm proper authorities might intentionally register unregistered domains to perform so called *sinkholing* – making some of the infected computers to connect to controlled servers instead of the original rogue ones.

To inspect HTTP traffic to those DGA domains clear the Wireshark view filter and apply the new one: ip.src == 10.0.0.2 && http


Filter: ip.src == 10.0.0.2 && http   Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
8	0.390487	10.0.0.2	10.0.0.1	HTTP	133	GET /project-feed/ HTTP/1.1
23	22.865046	10.0.0.2	10.0.0.1	HTTP	203	GET /pki/crl/products/CodeSignPCA.crl
74	164.466069	10.0.0.2	10.0.0.1	HTTP	325	GET / HTTP/1.1
85	166.316957	10.0.0.2	10.0.0.1	HTTP	342	GET / HTTP/1.1
96	168.048911	10.0.0.2	10.0.0.1	HTTP	342	GET / HTTP/1.1
107	169.847604	10.0.0.2	10.0.0.1	HTTP	336	GET / HTTP/1.1
118	171.682147	10.0.0.2	10.0.0.1	HTTP	343	GET / HTTP/1.1
129	173.402342	10.0.0.2	10.0.0.1	HTTP	341	GET / HTTP/1.1
140	175.251030	10.0.0.2	10.0.0.1	HTTP	343	GET / HTTP/1.1
151	176.987713	10.0.0.2	10.0.0.1	HTTP	343	GET / HTTP/1.1
162	178.804216	10.0.0.2	10.0.0.1	HTTP	339	GET / HTTP/1.1
173	180.591188	10.0.0.2	10.0.0.1	HTTP	341	GET / HTTP/1.1
184	182.397101	10.0.0.2	10.0.0.1	HTTP	344	GET / HTTP/1.1
195	184.226820	10.0.0.2	10.0.0.1	HTTP	342	GET / HTTP/1.1
206	186.052749	10.0.0.2	10.0.0.1	HTTP	343	GET / HTTP/1.1
217	187.940258	10.0.0.2	10.0.0.1	HTTP	337	GET / HTTP/1.1
228	189.757909	10.0.0.2	10.0.0.1	HTTP	341	GET / HTTP/1.1
239	191.616181	10.0.0.2	10.0.0.1	HTTP	339	GET / HTTP/1.1

Figure 95. HTTP traffic Wireshark filter.

The most interesting requests are GET / requests. To inspect HTTP headers and sent data right click on a few requests and choose *Follow TCP Stream* from the context menu. A new window with TCP stream should appear. After each click you will have to reapply the previous HTTP traffic filter because following the TCP stream automatically makes Wireshark change the view filter.

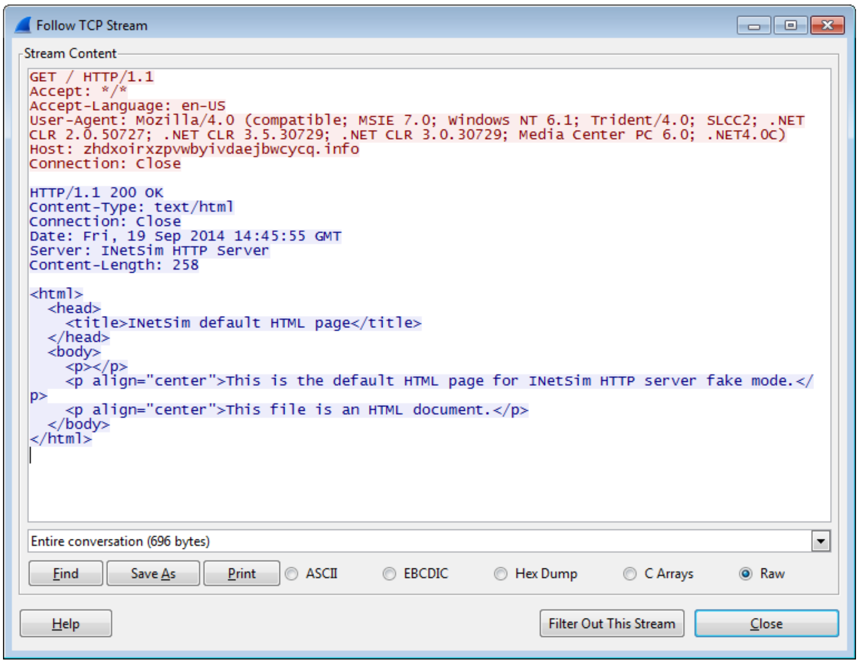


Figure 96. TCP Stream window in Wireshark.

In the TCP stream window the most interesting part is red text—the data sent by malware to the HTTP server. If the exercise was conducted with full access to the Internet, usually it would be also interesting to analyse real server replies (blue colour) – which might contain important information. In this case malware had only access to the network simulator – making server reply predictable and always the same.

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET
CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
Host: ofobeuylmnwctgtggvcrcamfehovu.com
Connection: Close
```

Figure 97. HTTP request to ofobeuylmnwctgtggvcrcamfehovu.com domain.

```
GET / HTTP/1.1
Accept: */*
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/4.0; SLCC2; .NET
CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)
Host: cieayptkrsldlmnvdqcmlin.com
Connection: Close
```

Figure 98. HTTP request to cieayptkrsldlmnvdqcmlin.com.

We see that among various HTTP requests only the *Host* value changes. It is important to note that the User-Agent string seems to be always the same. This might be used as a part of a network signature detecting this malware.

Now switch back to the Styx machine to do some analysis of the DGA domains.

One of the easy ways to get a list of DGA domain names is to use INetSim logs (other method would be to use Tshark tool present on Winbox machine). To do this, go to the INetSim results directory. In the report subdirectory there should be a single .txt file with report generated by INetSim.

Extracting list of DGA domains (on styx)

```
$ cd /lab/analyses/pz_7.exe/net_results/inetsim/report
$ ls
report.23733.txt
$ grep 'requested name` report.23733.txt | cut -d ' ' -f 12 >
domains.list
```

Now edit domains.list file and remove any domain that doesn't look like DGA domain e.g. *www.google.com, getgreenshot.org, etc.* – there shouldn't be too many such domains.

First check if there are any domains that appear multiple times.

Counting unique domains

```
$ cat domains.list | wc -l
152
$ cat domains.list | sort -u | wc -l
152
```

Both numbers should be the same meaning only unique names are present in the *domains.list* file.

Next check in what TLDs and ccTLDs are DGA domains.

Checking TLDs and ccTLDs

```
$ cat domains.list | cut -d '.' -f 2 | sort | uniq -c | sort -n
     20 net
     20 org
     21 info
     25 biz
     25 ru
     41 com
```

This means DGA domains are only in .net, .org, .info, .biz, .ru and .com domain with the last one having about twice as many entries as any other TLD.

It might be also useful to view average secondary-level domain name length (with TLD part stripped) to view if there is any pattern (e.g. all domains having the same length).

Checking DGA domain name length distribution

```
$ cat domains.list | cut -d '.' -f 1 | awk '{ print length }' | sort
-n | uniq -c
      1 18
      1 20
      8 21
      8 22
     14 23
     21 24
     22 25
     27 26
     18 27
     20 28
      8 29
      4 30
```

We see that most of the DGA domain names have length between 23 and 28 characters and almost all should have length between 18 and 30 characters.

Exercise:

Perform an analysis of the same sample for a second time, and try to answer the following questions (offline results available at /home/enisa/enisa/ex3/results/net1_2/net_results):

1. **Is the captured network traffic similar to the network traffic observed in the first analysis?**

   Yes, the captured traffic was similar. First there was a group of UDP datagrams and then malware started connecting to DGA domain names. In both cases there was also a single HTTP request to www.google.com host – after sending UDP datagrams finished. In all HTTP requests malware was using the same User-agent string.

2. **Was the malware sending UDP datagrams to the same IP addresses? What might this mean?**

   Yes. The malware was sending UDP datagrams to the list of the same IP addresses and to the same destination port each. This means that list of IP addresses was most likely hardcoded into the malware code.

3. **Was the UDP src port the same?**

   No. The source port of the UDP datagrams was different.

4. **Did the malware try to connect to the same domain names with HTTP protocol? What does it mean?**

   No. The malware was trying to connect to a completely different set of domains. This means that those domains were randomly generated (with the use of some algorithm known to the malware creator).

5. **Is a list of UDP addresses to which this sample sends datagrams a good network signature for detecting infections by this malware family?**

   No. It is not a very good indicator. Those addresses are constant for this particular sample. Other malware samples from the same family, that belong to a different botnet, will be sending UDP datagrams to different IP addresses.

6.3 HTTP traffic analysis

In this step sample l6XIE6749M.exe will be analysed. Capture the network traffic for this sample as it was described in the first step of this task. If during the analysis any dialog windows in Winbox appear accept them. It is assumed that result files will be stored in the directoru: /lab/analyses/ l6XIE6749M.exe/net_results/ .

In case there were any problems while performing the analysis, the result files can be also obtained from: /home/enisa/enisa/ex3/results/net2/net_results/

First go to the mitmproxy results directory and open mitmproxy logs.

Opening mitmproxy logs

```
$ cd /lab/analyses/l6XIE6749M.exe/net_results/mitmproxy/
$ ls
mitm.dump
$ mitmproxy -n --host -r mitm.dump
```

```
>> GET http://api.hostip.info/country.php
       ← 200 text/html 258B 255.57kB/s
   GET http://promos.fling.com/geo/txt/city.php
       ← 200 text/html 258B 88.4kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=2
       ← 200 text/html 258B 140.45kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=100
       ← 200 text/html 258B 315.26kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=99
       ← 200 text/html 258B 128.45kB/s
   GET http://goemqag.eu/rtce007.exe
       ← 200 x-msdos-program 24kB 21.21MB/s
   GET http://wabomiw.eu/jucheck.exe
       ← 200 x-msdos-program 24kB 17.49MB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=50
       ← 200 text/html 258B 444.7kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=22
       ← 200 text/html 258B 322.37kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=19
       ← 200 text/html 258B 94.98kB/s
   GET http://fpdownload.macromedia.com/get/flashplayer/update/current/install/install_all_win_ca
       b_ax_sgn.z
       ← 200 text/html 258B 201.73kB/s
   GET http://getgreenshot.org/project-feed/
[1/20]   [showhost]                                                                  ?:help
```

Figure 99. Mitmproxy window after reading log file.

To navigate through mitmproxy use arrow keys ([up], [down]). To view request details select request and press [Enter].

```
2014-09-19 17:22:56 GET http://api.hostip.info/country.php
                        ← 200 text/html 258B 255.57kB/s
Request                                           Response
Connection:        Keep-Alive
Accept:            */*
Accept-Language:   en-us
User-Agent:        Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Host:              api.hostip.info
No content

[1/20]   [showhost]                                                     ?:help q:back
```

Figure 100. Mitmproxy request details view.

In the request details view, to switch between request and server response use [Tab] key. At any point you can press 'q' key to go back.

After opening the mitmproxy logs obtained during the analysis, we see that there were several suspicious HTTP requests most likely done by the malware.

First two requests lead to the addresses:

- http://api.hostip.info/country.php
- http://promos.fling.com/geo/txt/city.php

```
>> GET http://api.hostip.info/country.php
       ← 200 text/html 258B 255.57kB/s
   GET http://promos.fling.com/geo/txt/city.php
       ← 200 text/html 258B 88.4kB/s
```

Figure 101. First two requests observed in mitmproxy.

The names of those URLs suggests they are used by malware to obtain geolocation data based on infected machine external IP address. Geolocation data is frequently used by malicious software to change its behaviour – some malware samples don't execute if started in certain countries while others might change their execution behaviour based on geolocation results (e.g. *ransomware* presenting messages in different languages).

Next there are six requests to *afferdls.cn* domain. Each of those requests has exact same headers and user-agent string. The only changing element is the value of GET parameter 'a'.

```
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=2
       ← 200 text/html 258B 140.45kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=100
       ← 200 text/html 258B 315.26kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=99
       ← 200 text/html 258B 128.45kB/s
   GET http://goemqag.eu/rtce007.exe
       ← 200 x-msdos-program 24kB 21.21MB/s
   GET http://wabomiw.eu/jucheck.exe
       ← 200 x-msdos-program 24kB 17.49MB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=50
       ← 200 text/html 258B 444.7kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=22
       ← 200 text/html 258B 322.37kB/s
   GET http://afferdls.cn/stat2.php?w=30000&i=000000000000000000000000009831c7cf&a=19
       ← 200 text/html 258B 94.98kB/s
```

Figure 102. Requests to afferdls.cn domain.

```
Request                                                     Response
Host:          afferdls.cn
User-Agent:    Opera/6 (Windows NT 6.1; ; LangID=409; x86)
Connection:    close
No content
```

Figure 103. afferdls.cn request headers.

Next we see a few requests for .exe files. In the analysed log there were 5 such requests:

- http://goemqag.eu/rtce007.exe (group 1)
- http://wabomiw.eu/jucheck.exe (group 1)
- http://alliswellintheuniverse.com/pRru4.exe (group 2)
- http://feyzmusteri.com/pAfy.exe (group 2)
- http://inzynieriawroclaw.soulhost.eu/yQQ1qD.exe (group 2)

The first two requests (group 1) were most likely done by a different malware module than requests from group 2. Requests from first group had different HTTP headers than requests from the second group. Also there is no negligible time difference between the executions of requests from each group.

```
2014-09-19 17:22:58 GET http://goemqag.eu/rtce007.exe
                        ← 200 x-msdos-program 24kB 21.21MB/s
Request                                                 Response
Host:   goemqag.eu
No content
```

Figure 104. Headers structure of requests from the first group.

```
2014-09-19 17:23:35 GET http://alliswellintheuniverse.com/pRru4.exe
                        ← 200 x-msdos-program 24kB 13.71MB/s
Request                                           Response
Host:         alliswellintheuniverse.com
Accept:       */*
Connection:   close
User-Agent:   Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
No content
```

Figure 105. Headers structure of requests from the second group.

We also know that the requested executables were executed on the Winbox system because a few popups appeared during the analysis informing that INetSim executable was executed (INetSim serves fake PE32 executable file when there is request for .exe file).

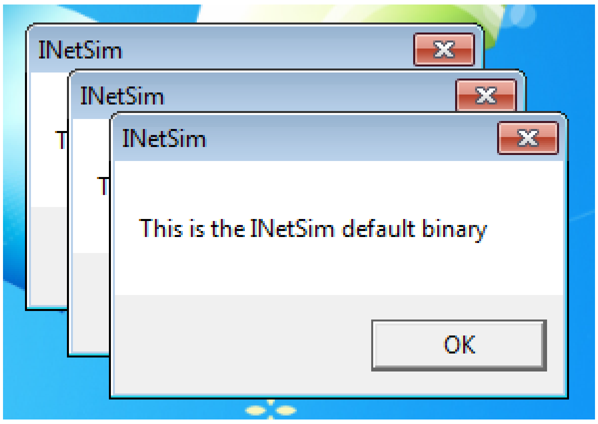


Figure 106. INetSim binary being executed on the Winbox machine.

Another interesting group of requests were the three requests to gate.php file:

- http://favoritepartner.com/ponyrtce/gate.php
- http://linercable.com/ponyrtce/gate.php
- http://biggestsetter.com/ponyrtce/gate.php

The characteristic gate.php filename suggests that those addresses are used by the malware to contact the C&C server. Next, let's view request details of one of those requests.

```
2014-09-19 17:23:34 POST http://favoritepartner.com/ponyrtce/gate.php
                         ← 200 text/html 258B 380.59kB/s
Request                                           Response
Host:               favoritepartner.com
Content-Length:     175
Connection:         close
Content-Type:       application/octet-stream
Content-Encoding:   binary
User-Agent:         Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
Raw
CRYPTED0.....?E..+..oX.Q...M.....i....fx....F.h7ZC.....2..B..*.)A../B....*...........\.0.R.......
Yh..XSM..@H.@A....m.N..1.{z.,.{%e0H...rPG....0....>#...,V..."".\_.,.|.........
```

Figure 107. Details of http://favoritepartner.com/ponyrtce/gate.php request.

There is some binary payload attached to the request. To ease viewing the binary payload, switch to hex view by pressing 'm' and then 'e'.

```
2014-09-19 17:23:34 POST http://favoritepartner.com/ponyrtce/gate.php
                        ← 200 text/html 258B 380.59kB/s
Request                                          Response
Host:               favoritepartner.com
Content-Length:     175
Connection:         close
Content-Type:       application/octet-stream
Content-Encoding:   binary
User-Agent:         Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)
Hex                                                                                  [m:Hex]
0000000000 43 52 59 50 54 45 44 30 94 8e 01 19 a5 3f 45 d2   CRYPTED0.....?E.
0000000010 ed 2b 1d ab 6f 58 ca 51 c0 c0 aa 4d ff e4 1f a0   .+..oX.Q...M....
0000000020 d5 69 d6 86 b8 18 66 78 98 c3 11 91 46 85 68 37   .i....fx....F.h7
0000000030 5a 43 0f d4 dc ed 14 32 f0 2e 42 eb dc 2a da 29   ZC.....2..B..*.)
0000000040 41 07 c5 2f 42 88 f6 c1 19 2a 14 00 80 f2 ee c2   A../B....*......
0000000050 04 9b 85 fa fe 5c 99 30 8c 52 fa cc 17 04 9c da   .....\.0.R......
0000000060 1e 59 68 db 8e 58 53 4d be a7 40 48 0e 40 41 99   .Yh..XSM..@H.@A.
0000000070 93 cc b8 6d d8 4e b2 17 31 b4 7b 7a fc 2c a3 7b   ...m.N..1.{z.,.{
0000000080 25 65 30 48 e5 83 08 72 50 47 cf cb c2 a1 30 ae   %e0H...rPG....0.
0000000090 c9 d2 92 3e 23 0e bd f1 2c 56 b4 0b a5 22 22 a7   ...>#...,V..."".
00000000a0 5f f2 2c 89 7c ad 93 f7 99 ea d9 96 2e 84 17      _.,.|..........

[13/20]  [showhost]                                                    ?:help q:back
```

Figure 108. Request payload in hex view.

When comparing this payload to payloads of other gate.php requests, we see that each request had exactly the same payload.

Exercise:

1. **Analyse the pcap file obtained in the same analysis of *I6XIE6749M.exe* sample. Is there any other suspicious network traffic besides http requests observed by MITMProxy?**

   Yes. There is suspicious non-http traffic in PCAP file.

   First there are a few UDP datagrams sent to 94.242.250.64 to port 53 (and seen by Wireshark as malformed DNS requests). This might be some covert channel created by the malware using port 53 to deceive system administrator.

No.	Time	Source	Destination	Protocol	Length	Info
33	1.788552	10.0.0.2	94.242.250.64	DNS	62	Unknown operation (15) 0x8889 [Malformed Packet]
36	1.788639	10.0.0.2	94.242.250.64	DNS	62	Unknown operation (15) 0x8889 [Malformed Packet]
45	1.800929	10.0.0.2	94.242.250.64	DNS	62	Unknown operation (15) 0x8889 [Malformed Packet]
107	15.007670	10.0.0.2	94.242.250.64	DNS	62	Unknown operation (15) 0x8889 [Malformed Packet]
114	16.293147	10.0.0.2	94.242.250.64	DNS	62	Unknown operation (15) 0x8889 [Malformed Packet]
119	16.297530	10.0.0.2	94.242.250.64	DNS	62	Unknown operation (15) 0x8889 [Malformed Packet]

Figure 109. Suspicious UDP traffic to port 53.

   Secondly there were a few TCP connection attempts to port 91.

Filter: tcp.port == 91 — Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
16	0.785332	10.0.0.2	10.0.0.1	TCP	66	49231 > mit-dov [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
17	0.785380	10.0.0.1	10.0.0.2	TCP	54	mit-dov > 49231 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
18	1.297251	10.0.0.2	10.0.0.1	TCP	66	49231 > mit-dov [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
19	1.297305	10.0.0.1	10.0.0.2	TCP	54	mit-dov > 49231 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
49	1.820686	10.0.0.2	10.0.0.1	TCP	62	49231 > mit-dov [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
50	1.820744	10.0.0.1	10.0.0.2	TCP	54	mit-dov > 49231 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

```
⊞ Frame 16: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: 08:00:27:66:25:2b (08:00:27:66:25:2b), Dst: 08:00:27:a7:7e:0e (08:00:27:a7:7e:0e)
⊞ Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.1 (10.0.0.1)
⊞ Transmission Control Protocol, Src Port: 49231 (49231), Dst Port: 91 (91), Seq: 0, Len: 0
```

Figure 110. Connection attempts to TCP port 91.

2. **During this exercise MITMProxy captured information about many HTTP connections. Were the addresses of all HTTP servers resolved by domain or were there any HTTP connections to hardcoded IP addresses?**

   The connection to http://afferdls.cn/stat2.php was done with a hardcoded IP address. There was no DNS request about *afferdls.cn* domain.

   In the current lab configuration if malware tries to connect to any domain, its address will be resolved to 10.0.0.1 (by INetSim fake DNS server). If the malware tries to connect to any service through a hardcoded IP address it will appear in the captured network traffic as a connection to an external IP address.

   This can be easily viewed in Wireshark:

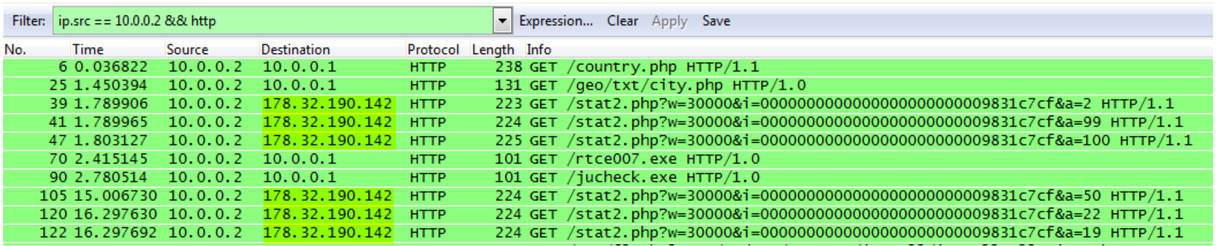


Figure 111. Connections to external IP address.

   It can be also viewed in MITMProxy if MITMProxy will be started without --*host* flag.

Starting mitmproxy without --host flag.

```
$ mitmproxy -n -r mitm.dump
```

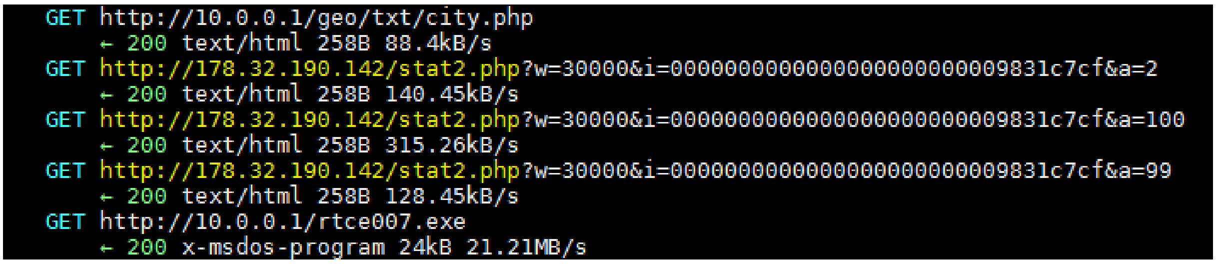


Figure 112. MITMProxy without --host flag.

6.4 Extra sample

As an extra exercise, students can analyse additional malware samples using the techniques known in this task. The extra sample name is: ejhct.bfg.exe. The sample can be found in /home/enisa/enisa/ex3/extra.

After analysis, students should have an open discussion to share their findings.

7 Task 4: Automatic analysis

After learning basic static analysis and dynamic analysis, students will be asked to perform automatic analysis using Cuckoo Sandbox to see what are advantages and disadvantages of such type of analysis. First, the students will upload the new sample to Cuckoo Sandbox and then they will analyse the results obtained.

To present all features of the Cuckoo Sandbox, a new malware sample, not analysed in previous tasks, will be used.

7.1 Sending sample to Cuckoo

First, start Cuckoo Sandbox with its web interface and API script as described in the first exercise *Building artifact handling and analysis environment*. Also make sure that INetSim is currently enabled. **NB: Cuckoo snapshot should in running state!**

Then start Viper (in *enisa* project space) and find invoice.exe sample. If there is no such sample it can be obtained from /home/enisa/enisa/ex3/samples directory.

```
enisa viper > find name invoice.exe
+---+-------------+----------------------+----------------------------------+------+
| # | Name        | Mime                 | MD5                              | Tags |
+---+-------------+----------------------+----------------------------------+------+
| 1 | invoice.exe | application/x-dosexec | a4f80b699b52c39da617a6614bb74d9f |      |
+---+-------------+----------------------+----------------------------------+------+
enisa viper > open -l 1
[*] Session opened on /opt/viper/projects/enisa/binaries/d/9/9/d/d99dfcdd814ef39468f69
12a8cf772f85eeeac285eb6c3187650eb9cd7833c79
enisa viper invoice.exe >
```

Figure 113. Finding invoice.exe sample in Viper

Then send sample to Cuckoo using the Viper *cuckoo* command.

```
enisa viper invoice.exe > cuckoo
{
    "task_id": 32
}
enisa viper invoice.exe >
```

Figure 114. Sending invoice.exe sample to the cuckoo analysis.

Then start Cuckoo web interface, switch to Recent tab and wait until the last analysis (md5: a4f80b699b52c39da...) will be completed and report generated.

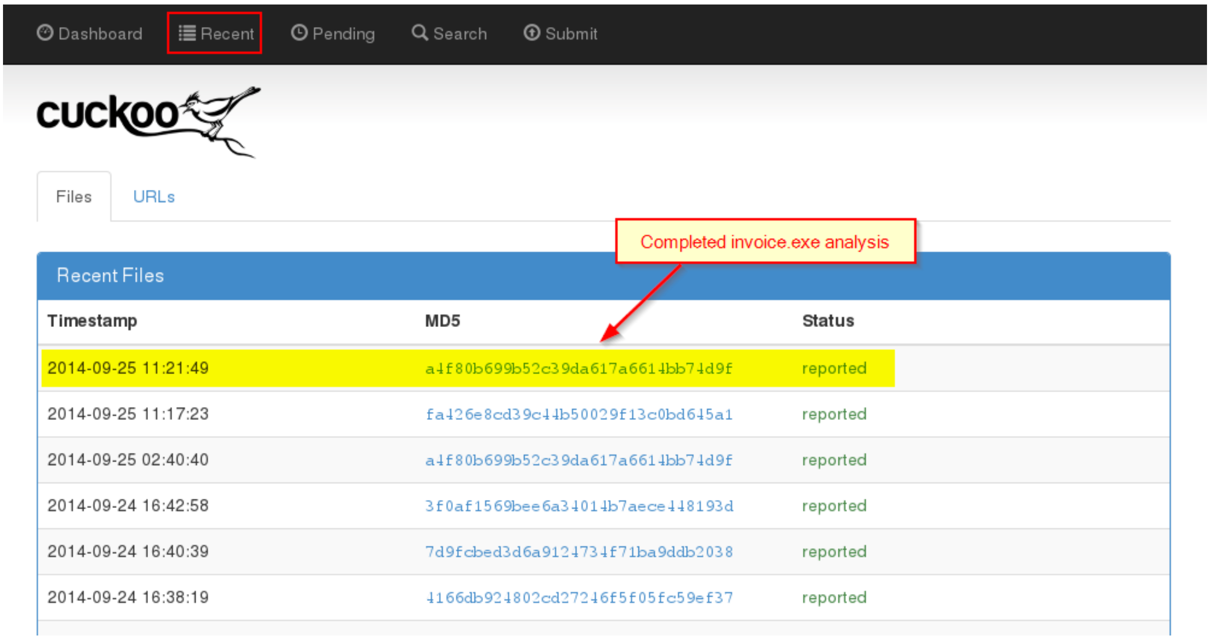
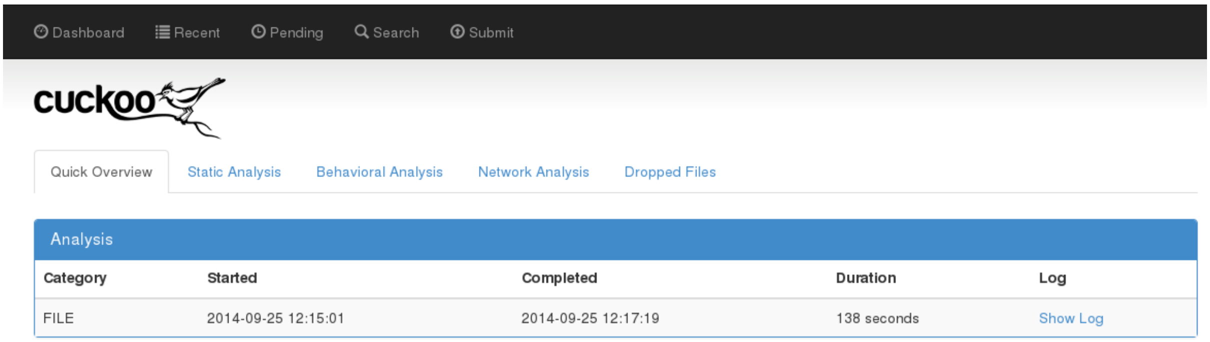


Figure 115. Completed invoice.exe analysis in Cuckoo Sandbox.

To view the analysis report click on md5 sum link.



Category	Started	Completed	Duration	Log
FILE	2014-09-25 12:15:01	2014-09-25 12:17:19	138 seconds	Show Log

Figure 116. Cuckoo report of the invoice.exe file.

Each Cuckoo report is divided into five areas: Quick Overview, Static Analysis, Behavioural Analysis, Network Analysis and Dropped Files. All of these areas will be briefly presented in the next steps of this task.

If there were any problems with starting Cuckoo Sandbox or sending sample to the analysis, offline analysis results can be obtained from /home/enisa/enisa/results/cuckoo1.

Offline results are in form of a saved webpage. To view them upload the results to the clean instance of the Winbox machine and open the result file (*cuckoo_invoice.htm*) in a web browser. Then proceed with the analysis as it is described in the next step.

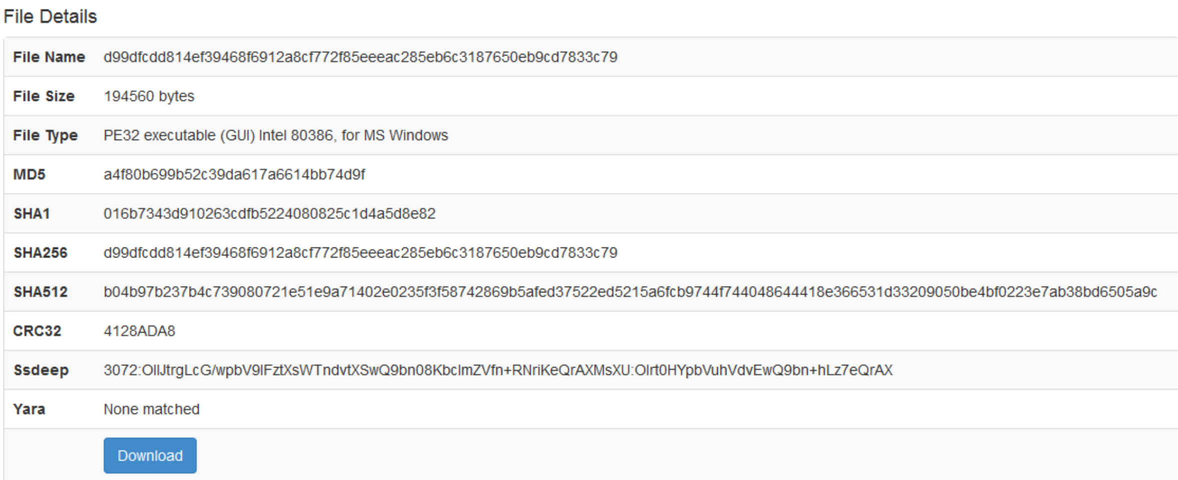
Sending offline cuckoo results to Winbox

```
$ lab-sendfile /home/enisa/enisa/results/cuckoo1
```

7.2 Cuckoo Sandbox results

The first area of the Cuckoo Sandbox report is *Quick Overview* giving brief information about analysed sample and its behaviour.

At the top of the *Quick Overview* there is *File Details* section presenting a sample file name, checksums as well as any detected signatures. An interesting thing to notice is that if the sample is uploaded to Cuckoo Sandbox using Viper, the original sample file name is changed to its SHA256 sum value. This is not the case when the sample is uploaded by web interface or Cuckoo scripts.


File Details

File Name	d99dfcdd814ef39468f6912a8cf772f85eeeac285eb6c3187650eb9cd7833c79
File Size	194560 bytes
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
MD5	a4f80b699b52c39da617a6614bb74d9f
SHA1	016b7343d910263cdfb5224080825c1d4a5d8e82
SHA256	d99dfcdd814ef39468f6912a8cf772f85eeeac285eb6c3187650eb9cd7833c79
SHA512	b04b97b237b4c739080721e51e9a71402e0235f3f58742869b5afed37522ed5215a6fcb9744f744048644418e366531d33209050be4bf0223e7ab38bd6505a9c
CRC32	4128ADA8
Ssdeep	3072:OlIJtrgLcG/wpbV9lFztXsWTndvtXSwQ9bn08KbclmZVfn+RNriKeQrAXMsXU:Olrt0HYpbVuhVdvEwQ9bn+hLz7eQrAX
Yara	None matched
	Download

Figure 117. File Details section of the Cuckoo report.

The following section is presenting what hosts the malware connected to, and what domains it was querying? In this case we can see two suspicious domains: *angelescitypattaya.com* and *pattayasuay.com*.

Hosts

IP
8.8.8.8
239.255.255.250

Domains

Domain	IP
teredo.ipv6.microsoft.com	
dns.msftncsi.com	
angelescitypattaya.com	
pattayasuay.com	
crl.microsoft.com	
ocsp.msocsp.com	

Figure 118. List of hosts and domains from the Cuckoo report.

The summary section below is presenting list of files, registry keys and mutexes which malware accessed during the analysis (created, read or written).

Summary

Files Registry Keys Mutexes

```
C:\Windows
C:\Windows\
C:
MountPointManager
C:\Users\ENISA\AppData\Local\Temp\d99dfcdd814ef39468f6912a8cf772f85eeeac285eb6c3187650eb9cd7833c79
C:\Users\ENISA\AppData\Roaming\Ozho\unweh.exe
C:\Users\ENISA\AppData\Roaming\Tymicy
C:\Device\HarddiskVolume2\Users\ENISA\AppData\Roaming\
C:\Users\ENISA\AppData\Local\Temp
C:\Users\ENISA\AppData\Roaming\Tymicy\igruo.duo
C:\Users\ENISA\AppData\Roaming\Ixyxm
C:\Users\ENISA\AppData\Roaming\Ixyxm\ydpai.von
```

Figure 119. Fragment of the list of accessed files.

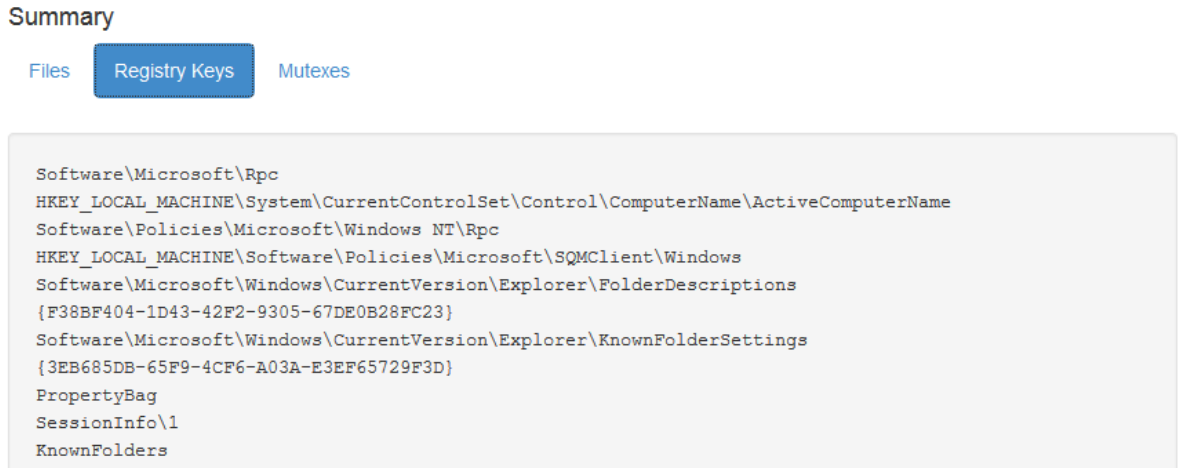


Figure 120. Fragment of the list of accessed registry keys.

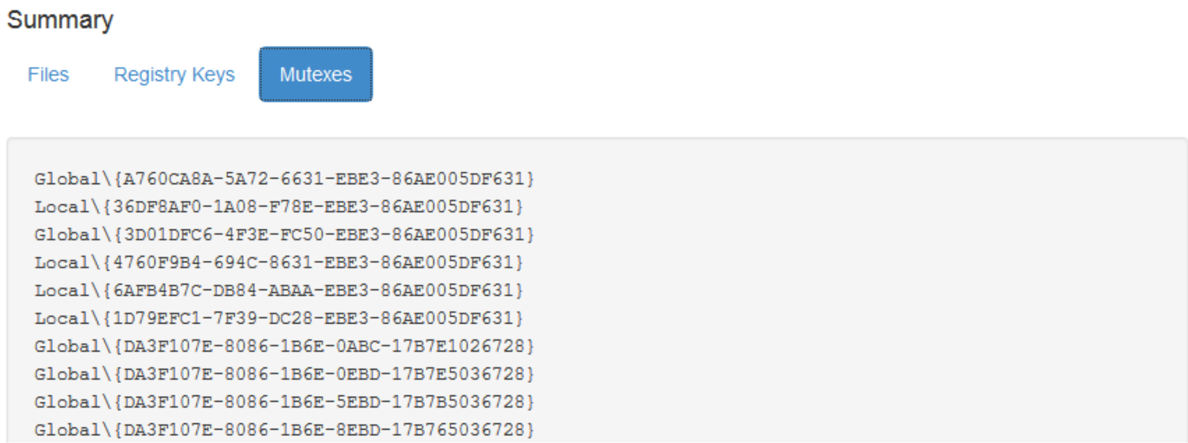


Figure 121. Fragment of the list of accessed mutexes.

7.3 Static Analysis results

Switch to the *Static Analysis* area of the Cuckoo Sandbox report. This section contains information about static analysis findings. Additionally *Static Analysis* is divided into three subsections: static analysis, strings and antivirus.

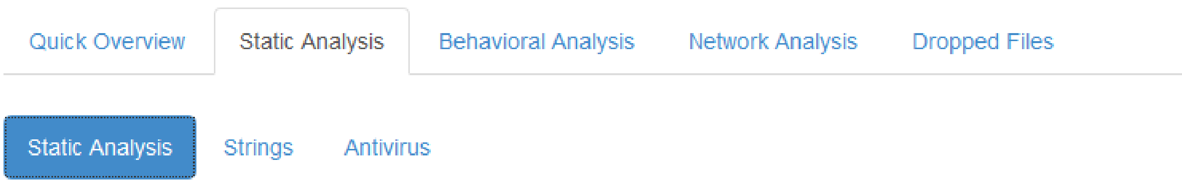


Figure 122. Static Analysis section with three subsections.

The Static Analysis subsection starts with the Version Info structure – structure which is typically attached to the executable file as an additional resource[10]. The aim of this structure is to give information about the executable version number, operating system, description, as well as the original file name.

[10] http://msdn.microsoft.com/en-us/library/windows/desktop/aa381058%28v=vs.85%29.aspx

Version Infos

LegalCopyright	\xa9 2003 Gelepu Erovoz. Osahany Agatul Eleliny.
dX8r6WrHnFP	UbqTEf2b2nqR
h83w2oFDe3CRoKi	fP1aC6UPaooDatDNrJtq
thd2BBGBxqD3w8wg	dbOtMQbFhVgLSBq
AuK4phoSjGe5Wk	xNLCrxAVRSKOXEDS
peBMItR6LwTKIpKYbqw8	q7dqTstM5U7w
OESxJwGaUTnX	rDPfQkcTX7Vk

Figure 123. Fragment of the Version Info structure of the analysed sample.

In this case we see that Version Info structure is filled with random strings – this is not a typical situation.

Below Version Info structure there is a list of PE sections found in executable file.

Sections

Name	Virtual Address	Virtual Size	Size of Raw Data	Entropy
6\xae[\xb7\xc1-\x81\xd6	0x00001000	0x00035000	0x00000000	0.0
\x87\x17j\x97\xeb\xf1\x0b\x11	0x00036000	0x0002f000	0x0002e800	7.99661984982
.rsrc	0x00065000	0x00001000	0x00000c00	3.37805487255

Figure 124. List of sections in analysed binary file.

We can see that the first two sections have some random names. Moreover the second section has very high entropy (7.99/8.00) while the first section has no raw data on disk and large virtual size. This is a clear indicator that this sample was packed.

The sections below list the sample imports lists. We see that malware imports only a few functions from three libraries. This confirms our suspicion that this sample was packed.

Imports

Library KERNEL32.DLL:
- `0x465a0c` LoadLibraryA
- `0x465a10` GetProcAddress
- `0x465a14` VirtualProtect
- `0x465a18` VirtualAlloc
- `0x465a1c` VirtualFree
- `0x465a20` ExitProcess

Library aDVaPI32.dll:
- `0x465a28` CryptSetProvParam

Library UsER32.DLL:
- `0x465a30` SetPropA

Figure 125. Imports list of the sample.

Next, switch to Strings subsection which contains strings found in sample file. As expected from the packed file there aren't too many meaningful strings for this sample.

Static Analysis | **Strings** | Antivirus

```
!This program cannot be run in DOS mode.
aw?%.!
tgs#/*
0P:gj8
B/x%/Q
\I:?zy
(q#@:5{
Nn(;e,
S9U;<Q
]y{u%T
_U4F5^8
#@Y|}C!{c
.9(W,D
```

Figure 126. Fragment of the Strings subsection.

Depending on whether there was internet access on the Styx machine, in the next Antivirus subsection there will be a list of Virustotal results for the analysed file (if there was no Internet access this subsection will be empty).

Static Analysis Strings **Antivirus**

Antivirus	Signature
Bkav	W32.AppdataZano.Trojan
MicroWorld-eScan	Trojan.GenericKDV.959150
nProtect	Trojan.GenericKDV.959150
CMC	Trojan-Spy.Win32.Zbot!O
CAT-QuickHeal	TrojanPWS.Zbot
McAfee	Artemis!A4F80B699B52
Malwarebytes	Trojan.Agent.RSRVGen
K7AntiVirus	Trojan (0040f3081)
K7GW	Trojan (0040f3081)

Figure 127. Fragment of the Virustotal results list for the analysed sample.

7.4 Behavioural Analysis results

Behavioural Analysis results section contains information on what malicious processes were running during the analysis. It lists processes started by the malware as well as processes to which the malware injected its code.

At the top there is a process tree of the malware's processes.

- **d99dfcdd814ef39468f6912a8cf772f85eeeac285eb6c3187650eb9cd7833c79** 1540
  - **unweh.exe** 580
  - **cmd.exe** 3692
- **taskhost.exe** 1876
- **Dwm.exe** 1740
- **Explorer.EXE** 1764
  - **FileZilla Server Interface.exe** 1412
  - **Greenshot.exe** 852

Figure 128. Malware processes process tree.

On this list we see that the malware sample (*d99dfc…*, pid:1540) created two new processes: unweh.exe and cmd.exe. There is also a group of other processes involved in malware activity: taskhost.exe, Dwm.exe, Explorer.EXE, FileZilla Server Interface.exe and Greenshot.exe to which malware might have injected some code.

In the process tree below, there is an API calls list for each traced process. To switch between processes click on tabs with process names. It is also possible to filter API calls by clicking on the chosen calls type.

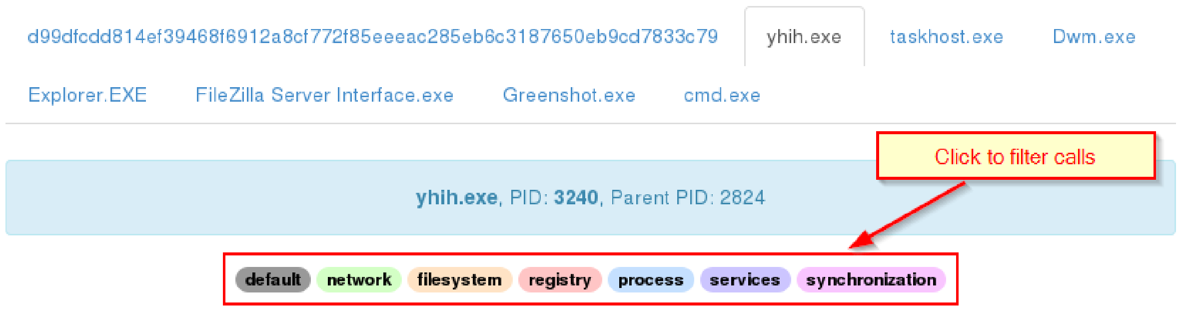


Figure 129. API calls list.

Each observed API call consists of timestamp when it was observed, its name, arguments, status, return value and information whether it was repeated.

Time	API	Arguments	Status	Return	Repeated
2014-09-24 14:02:30,144	**NtOpenDirectoryObject**	DirectoryHandle: 0x00000064 DesiredAccess: 15 ObjectAttributes: C:\Sessions \1\BaseNamedObjects	success	0x00000000	
2014-09-24 14:02:31,582	**NtOpenFile**	ShareAccess: 3 FileName: C:\Windows DesiredAccess: 0x00100080 FileHandle: 0x00000000	failed	3221225658	

Figure 130. Two example API calls.

By tracing the calls made by each process, it is possible to find out information about some of the malicious code’s functionality. Unfortunately due to the usually large number of observed calls it is a rather time consuming task.

Due to the structure of the results page, API calls won’t be available for students using offline results file *cuckoo_invoice.htm*. Students using offline results can still view API calls using slightly older Cuckoo report format by opening the second file – *cuckoo_invoice2.htm*.

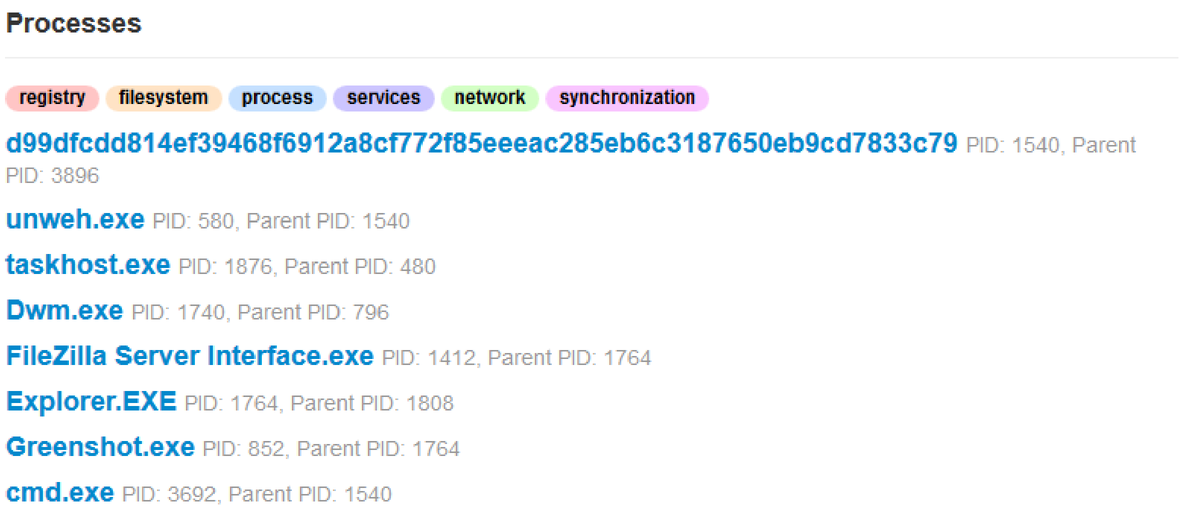


Figure 131. Older format of API calls list in cuckoo_invoice2.htm.

7.5 Network Analysis results

The Network Analysis section includes information about network traffic observed during the analysis. Currently, the detected traffic types are DNS requests, HTTP traffic, ICMP packets and IRC protocol. It is also possible to directly download the PCAP file with all of the detected traffic for further inspection.

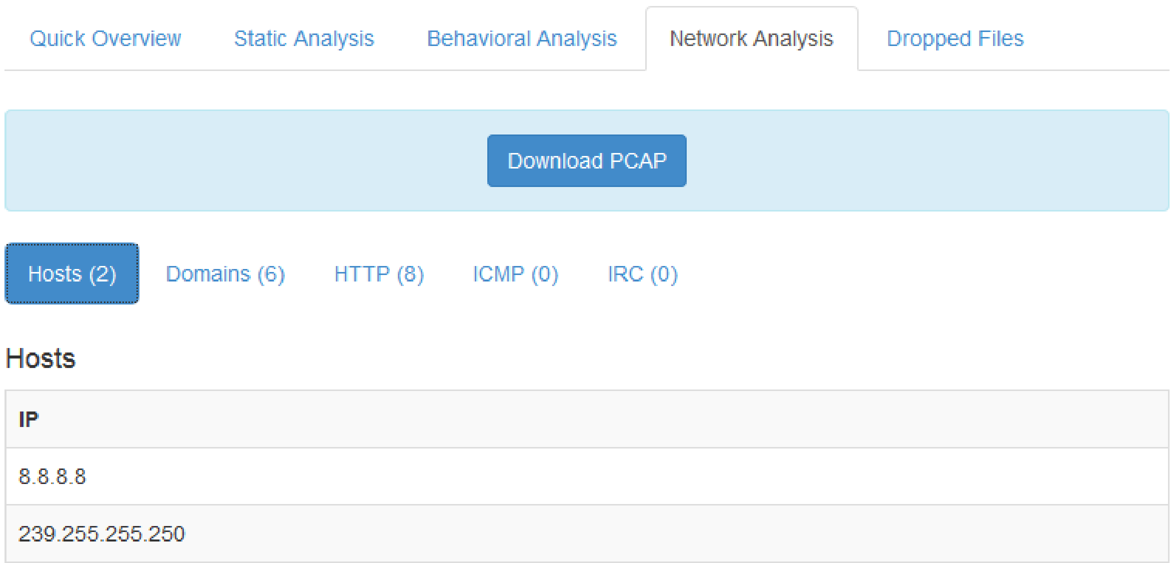


Figure 132. Network traffic analysis section.

Hosts and Domains were already listed in the *Quick Overview* section. The only other recognized traffic are eight HTTP requests.

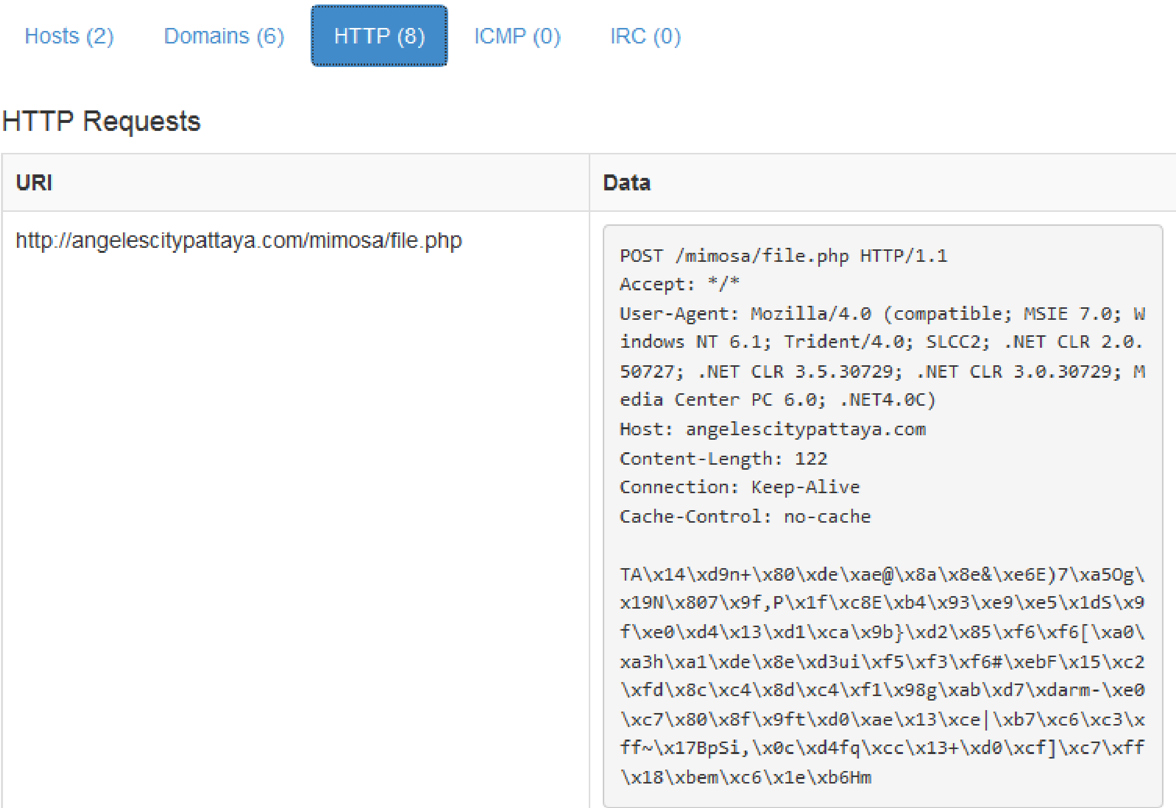


Figure 133. HTTP requests subsection.

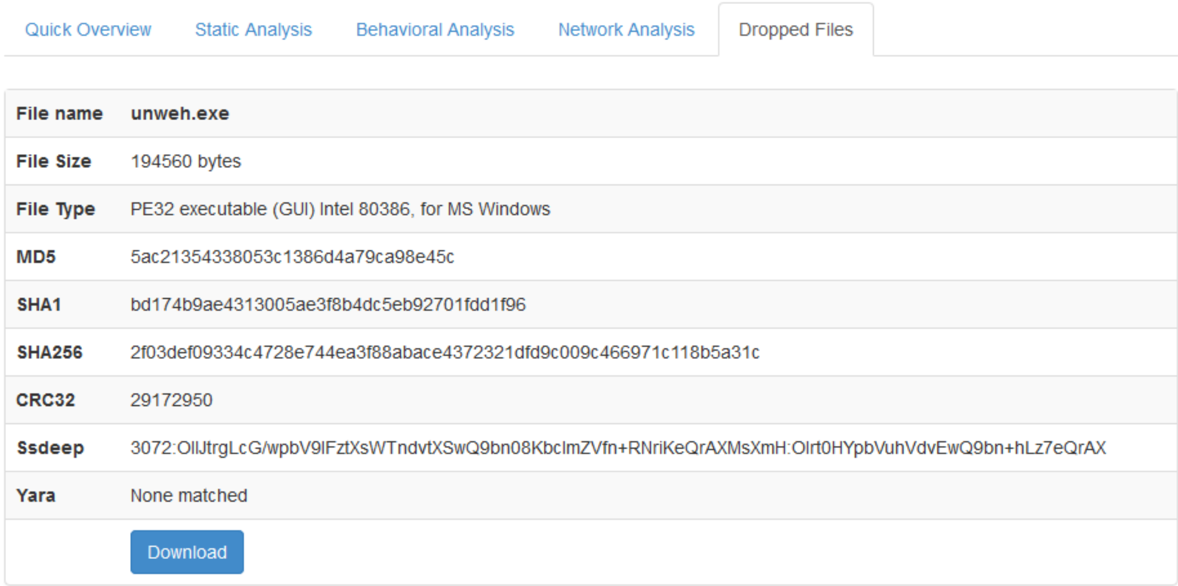
In the HTTP requests subsection we can see that the malware was doing multiple suspicious HTTP POST requests to file.php. In each such request the same User-agent string was used. There was also variable length POST data attached, different for each request.

In total there were six requests to file.php to two unique URIs:

- http://angelescitypattaya.com/mimosa/file.php
- http://pattayasuay.com/dkp/file.php

7.6 Analysing list of dropped files

The last section of the Cuckoo report (*Dropped Files*) contains a list of files that were observed to be created during the analysis.



File name	**unweh.exe**
File Size	194560 bytes
File Type	PE32 executable (GUI) Intel 80386, for MS Windows
MD5	5ac21354338053c1386d4a79ca98e45c
SHA1	bd174b9ae4313005ae3f8b4dc5eb92701fdd1f96
SHA256	2f03def09334c4728e744ea3f88abace4372321dfd9c009c466971c118b5a31c
CRC32	29172950
Ssdeep	3072:OlIJtrgLcG/wpbV9lFztXsWTndvtXSwQ9bn08KbclmZVfn+RNriKeQrAXMsXmH:Olrt0HYpbVuhVdvEwQ9bn+hLz7eQrAX
Yara	None matched
	Download

Figure 134. Dropped Files section.

For each dropped file there is a standard file details table containing the file name, type, and group of cryptographic hashes. Additionally each file can be downloaded to the local machine.

In this analysis, Cuckoo Sandbox detected the creation of the following files:

- unweh.exe (executable)
- d99dfcdd814ef39468f6912a8cf772f85eeeac285eb6c3187650eb9cd7833c79 (executable)
- tmpfd5ba7aa.bat (DOS batch file)
- igruo.duo (unknown data)
- file[1].htm (HTML document/text)
- file[2].htm (HTML document/text)

7.7 Extra analyses

As an extra exercise, students can analyse samples from previous tasks using Cuckoo Sandbox. Then they should compare the Cuckoo Sandbox results with their previous findings. Most of the results should be similar to the previous ones. For a few samples Cuckoo Sandbox might fail during the static analysis or report some errors. This is caused by some obfuscation techniques used by malware or some other non-standard behaviour.

8 Exercise summary

During the exercise students have learnt basic principles of malicious artifacts analysis. After a proper theoretical introduction, the students had the opportunity to test their skills by analysing live malware samples.

At the beginning of the exercise the students were introduced to the fundamentals of malicious code analysis. In this part, the students learnt various types of analyses, their application, strong and weak points, and when to use each of them. After that, the participants learnt basic security precautions involving the execution of malware samples in a controlled environment.

During the basic static analysis, students had the opportunity to search for indicators of the malicious functionality in the sample files provided. First they scanned the sample for the patterns of well-known packers and protectors, then they analysed a list of strings extracted from the file. After the string analysis, the participants analysed various headers in the PE structure (import tables, file resources). Finally, the students scanned a sample malware for any embedded objects with well-known file types.

After the static analysis, the students performed basic behavioural analysis of the provided sample. During this analysis they searched for any changes in the operating system that might indicate malicious code functionality and purpose. After that, that operating system was scanned using the GMER tool to search for any indicators of rootkit activity.

During the network analysis, the participants executed the samples provided, and captured the network traffic. The samples were executed in an isolated environment. To simulate basic network services, INetSim tool was used. Then, using the captured traffic, students searched for well-known malicious network traffic patterns.

The last type of analysis performed was an automatic analysis. During this analysis, students used the Cuckoo Sandbox appliance previously configured in the exercise *Building artifact handling and analysis environment.* The purpose of this analysis was to let students compare the results obtained in the automatic analysis with the results from non-automatic analyses.

9 References

1. A Study of the Packer Problem and Its Solutions
   http://www.ecsl.cs.sunysb.edu/tr/TR237.pdf (accessed 15. October 2014)
2. Visual Studio 2013 Update 3 http://www.visualstudio.com/ (accessed 15. October 2014)
3. What is a DLL? http://support.microsoft.com/kb/815065 (accessed 15. October 2014)
4. Base64 http://en.wikipedia.org/wiki/Base64 (accessed 15. October 2014)
5. Borland Delphi http://www.on-time.com/rtos-32-docs/rttarget-32/programming-manual/compiling/borland-delphi.htm (accessed 15. October 2014)
6. Advanced Techniques – Using Process Explorer
   http://www.microsoft.com/security/sir/strategy/default.aspx#!malwarecleaning_explorer
   (accessed 15. October 2014)
7. VERSIONINFO resource http://msdn.microsoft.com/en-us/library/windows/desktop/aa381058%28v=vs.85%29.aspx (accessed 15. October 2014)

ENISA
European Union Agency for Network and Information Security
Science and Technology Park of Crete (ITE)
Vassilika Vouton, 700 13, Heraklion, Greece

Athens Office
1 Vass. Sofias & Meg. Alexandrou
Marousi 151 24, Athens, Greece

PO Box 1309, 710 01 Heraklion, Greece
Tel: +30 28 14 40 9710
info@enisa.europa.eu
www.enisa.europa.eu