



# **Artifact analysis fundamentals**

Artifact analysis training material

November 2014





European Union Agency for Network and Information Security



# 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 <u>www.enisa.europa.eu</u>.

# Authors

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

# Contact

For contacting the authors please use <u>cert-relations@enisa.europa.eu</u>

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.

<sup>&</sup>lt;sup>1</sup> Klaus Möller, and Mirko Wollenberg

<sup>&</sup>lt;sup>2</sup> Mirosław Maj, Tomasz Chlebowski, Krystian Kochanowski, Dawid Osojca, Paweł Weżgowiec, and Adam Ziaja

<sup>&</sup>lt;sup>3</sup> Michael Potter, Alan Robinson, and Don Stikvoort



### Legal notice

Notice must be taken that this publication represents the views and interpretations of the authors and editors, unless stated otherwise. This publication should not be construed to be a legal action of ENISA or the ENISA bodies unless adopted pursuant to the Regulation (EU) No 526/2013. This publication does not necessarily represent state-of the-art and ENISA may update it from time to time.

Third-party sources are quoted as appropriate. ENISA is not responsible for the content of the external sources including external websites referenced in this publication.

This publication is intended for information purposes only. It must be accessible free of charge. Neither ENISA nor any person acting on its behalf is responsible for the use that might be made of the information contained in this publication.

#### **Copyright Notice**

© European Union Agency for Network and Information Security (ENISA), 2014

Reproduction is authorised provided the source is acknowledged.



# **Table of Contents**

1	General description	2
2	Introduction	3
2.1	Malware analysis fundamentals	3
2.2	Various approaches to malware analysis	3
2.3	Safety precautions	4
2.4	Exercise remarks	5
3	Tools overview	5
3.1	Static analysis tools	5
3.2	Dynamic analysis tools	6
3.3	Network analysis tools	6
3.4	Automatic analysis tools	7
4	Task 1: Basic static analysis	7
4.1	Sending sample to the analysis.	7
4.2	Detecting packers and protectors	8
4.3	Strings extraction and analysis	11
4.4	PE structure and headers analysis	15
4.5	Import table analysis	17
4.6	PE resources analysis	21
4.7	Searching for embedded objects	23
4.8	Finishing analysis	23
4.9	Extra samples	24
5	Task 2: Behavioural analysis	24
5.1	Analysis remarks	25
5.2	Preparing analysis	25
5.3	Executing malware sample	29



5.4	Process Explorer analysis	31
5.5	Regshot analysis	35
5.6	Process Monitor analysis	36
5.7	Searching for rootkit artifacts	42
5.8	Finishing analysis	43
5.9	Extra samples	43
6	Task 3: Network analysis	44
6.1	Network traffic capture and log acquisition	44
6.2	P2P and DGA traffic	46
6.3	HTTP traffic analysis	55
6.4	Extra sample	60
7	Task 4: Automatic analysis	60
7.1	Sending sample to Cuckoo	61
7.2	Cuckoo Sandbox results	62
7.3	Static Analysis results	64
7.4	Behavioural Analysis results	67
7.5	Network Analysis results	69
7.6	Analysing list of dropped files	70
7.7	Extra analyses	71
8	Exercise summary	71
9	References	72



Main Objective	Present the trainees malicious artifact analysis fundar various types of analyses. Present how to safely execute sus in the controlled environment along with most import precautions. Teach the trainees how to perform behavioural, network and automatic analyses – what tools what to look for, what can be found. Give the trainees the to use various popular tools during the analyses and let what tools are best suited for different type of analy common malicious software behaviours and patterns – v later used to create proper signature.	spicious code tant security basic static, can be used, copportunity them decide ses. Present
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	
	Introduction to the exercise and tools overview	1.0 hours
	Task 1: Basic static analysis	1.5 hours
Time Schedule	Task 2: Behavioural analysis	2.0 hours
Time Schedule	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 me 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.

	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.
Basic static analysis	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. <u>http://www.ntcore.com/exsuite.php</u>



 Resource Hacker – popular tool to view, modify, rename, add, delete and extract resources in 32bit & 64bit Windows executables and resource files.
 http://www.apgusi.com/resourcebacker/

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. <a href="http://www.mcafee.com/us/downloads/free-tools/bintext.aspx">http://www.mcafee.com/us/downloads/free-tools/bintext.aspx</a>
- 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
  <a href="http://technet.microsoft.com/en-US/sysinternals/bb896653">http://technet.microsoft.com/en-US/sysinternals/bb896653</a>
- **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.

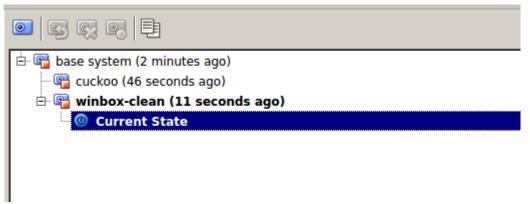
In this task the students will try to determine basic malware functionalities with a help from the trainer and the purpose of the task is to look for any special features which might be later used to create the file signature.

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.



#   Name   Mime	MD5	Tags
	7a0938b535f1bbd7a85065249bbbefd1	
+	//a0938053571000/a85065249000e701 +	+

Figure 2. Finding aop.exe sample in Viper.

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



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).

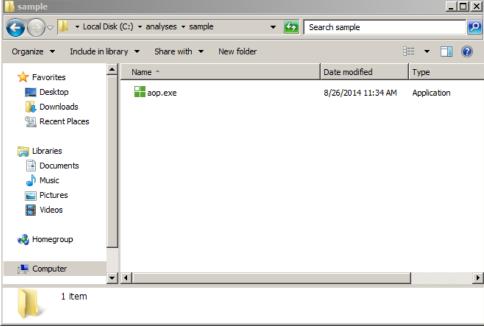


Figure 4. Malware sample after uploading to Winbox machine.

# 4.2 Detecting packers and protectors

Malware samples are often protected by so called packers and protectors<sup>4</sup>. 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

<sup>&</sup>lt;sup>4</sup> <u>http://www.ecsl.cs.sunysb.edu/tr/TR237.pdf</u>



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:

November 2014

eiD v0.	95		
File: C:\an	alyses\sample\ao	p.exe	
Entrypoint:	00024A50	EP Section	: UPX1 >
File Offset:	0000BE50	First Bytes	: 60,BE,00,90 >
Linker Info:	6.0	Subsystem	Win32 GUI >
UPX 0.89.6	- 1.02 / 1.05 - 2.	90 -> Markus & Laszlo	
Multi Scan	Task Viewe	r <u>O</u> ptions <u>A</u> l	oout E <u>x</u> it
Stay on	top		>> ->

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:

属 Exe	info PE - ver.0.0.3.5 Beta by A.S.L -	748+4 sig 2014.02.24	_ 🗆 X
	<u>File</u> : aop.exe		<u>H</u> R
	Entry Point : 00024A50 00 <	EP Section : UPX1	> S
	File Offset : 0000BE50	First Bytes : 60.BE.00.90.41	<u>i</u> Plug
(2)	Linker Info: 6.00	SubSystem : Windows GUI	
B	File Size : 00000CE00h < N	Overlay : NO 00000000	Options
nfa	Image is 32bit executable	RES/OVL: 4 / 0 % 2014	Exit
1	UPX -> Markus & Laszlo ver. [ 3.08 ] Lamer Info - Help Hint - Unpack info unpack "upx.exe -d" from http://upx.s	0 ms.	

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.



November 2014

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.

npacked.exe c:\analyses\s U]	ample\aop ltimate Pa Copyrigh	cker for eXec t (C) 1996 -	
File size	Ratio	Format	Name
135168 <- 52736	39.02%	win32/pe	aop_unpacked.exe
Unpacked 1 file.			
C:\tools\Portable version	ı\upx391₩\	upx391w>	~

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.

🕮 PEiD v0.95	
File: C:\analyses\sample\ao	p_unpacked.exe
Entrypoint: 000154EC File Offset: 000154EC	EP Section:     .text     >       First Bytes:     55,88,EC,6A     >
Linker Info: 6.0	Subsystem: Win32 GUI >
Microsoft Visual C++ 6.0 <u>M</u> ulti Scan <u>Task Viewer</u> ✓ Stay on top	<u>Options</u> <u>A</u> bout <u>Exit</u> »» ->

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)<sup>5</sup>.

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

# 4.3 Strings extraction and analysis

November 2014

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.

File to scan C:	File to scan C:\analyses\sample\aop_unpacked.exe			
Advanced view			Time taken : 0.016 secs	Text size: 8629 bytes (8.4
File pos	Mem pos	ID	Text	
A 0000000004D	00000040004D	0	This program cannot be run i	n DOS mode.
A 000000000E7	0000004000E7	0	}Rich	
A 00000000200	000000400200	0	.text	
A 00000000228	000000400228	0	.rdata	
A 0000000024F	00000040024F	0	@.data	
A 00000000278	000000400278	0		
<b>A</b> 0000000019B2	0000004019B2	0	\$SUVW	
A 00000001BE0	000000401BE0	0	L\$\$Pi	
A 000000001D1F	000000401D1F	0	L\$8Pi	
A 00000002925	000000402925	0	ป<\น8	
A 000000002CF2	000000402CF2	Ō	QPPPPPS	
A 000000002D3C	000000402D3C	0	<at;<bt7< td=""><td></td></at;<bt7<>	
A 00000002FF6	000000402FF6	0	L\$(QR	
A 200000000000	000000400004	<u> </u>	DOWN	

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.

<sup>&</sup>lt;sup>5</sup> See <u>http://www.visualstudio.com/</u> for more details on this development environment.



<b>7</b> Save text to file					×
Compu	uter 🔹 Local Disk (C:) 👻 analyses 👻 results 👻	·	Search results		2
Organize 🔻 New folde	r				0
🧮 Desktop	Name ^	Date modified	Туре	Size	
ib Downloads 🖳 Recent Places	Screenshots	8/26/2014 11:56 AM	File folder		
Cibraries					
<ul> <li>Music</li> <li>Pictures</li> <li>Videos</li> </ul>					
	•				
File <u>n</u> ame: a	pp_unpacked_strings.txt				-
Save as type: Tx	t files (*.txt)				•
Hide Folders			Save	Cancel	

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
<b>A</b> 00000001930A	00000041930A	0	SHLWAPI.dll
<b>A</b> 000000019316	000000419316	0	USER32.dll
A 000000019321	000000419321	0	WININET.dli 📐
A 00000001932D	00000041932D	0	WS2_32.dli 🛛 📉
A 000000019338	000000419338	0	WTSAPI32.dll
A 000000019348	000000419348	0	IstropyA
A 000000019352	000000419352	0	SetÉvent
A 00000001935C	00000041935C	0	InterlockedExchange
A 000000019372	000000419372	0	Cancello
A 00000001937C	00000041937C	0	DeleteFileA
A 00000001938A	00000041938A	0	GetLastError
A 000000019398	000000419398	0	CreateDirectoryA
<b>A</b> 0000000193AA	0000004193AA	0	GetFileAttributesA
A 0000000193BE	0000004193BE	0	IstrienA

Figure 12. Windows functions and library names.

Here we can see a fragment of the DLL names<sup>6</sup> 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.

<sup>&</sup>lt;sup>6</sup> <u>http://support.microsoft.com/kb/815065</u>



Artifact analysis fundamentals

Artifact analysis training material

November 2014

A 0000	D001B12C	00000041B12C	0	%s\*.*
A 0000	DOO1B134	00000041B134	0	FindFirstFileA
A 0000	DOO1B144	00000041B144	0	LocalReAlloc
A 0000	DOO1B154	00000041B154	0	FindNextFileA
A 0000	DOO1B164	00000041B164	0	%s\%s
A 0000	DOO1B16C	00000041B16C	0	%s%s%s
A 0000	D001B174	00000041B174	0	%s%s*.*
		Eiguro 12	-	ath matching expressions found in strings list

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.

<b>A</b> 00000001B180	00000041B180	0	system\cURRENTcONTROLsET\sERVICES\%s
A 00000001B1A8	00000041B1A8	0	OpenSCM anagerA
A 00000001B1C0	00000041B1C0	0	system\cURRENTcONTROLsET\sERVICES\
		- Figure 1	4. 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).

<b>A</b> 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/pipeg, application/x-sh		
A 00000001B4AC	00000041B4AC	0	Accept-Language: zh-cn		
A 00000001B4C8	00000041B4C8	0	Accept-Encoding: gzip, deflate		
A 00000001B4EA	00000041B4ፑጲ	0	User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)		
A 00000001B532	00000041B53&	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:		
<b>A</b> 00000001B586	00000041B586	0	Cache-Control: no-cache		
A 00000001B5A0	00000041B5A0	0	Host:		
A 00000001B5A8	00000041B5A8	0	HTTP/1.1		
A 00000001B5B6	000000418586	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
		Figu	are 17. Strings with batch command.

Typical batch command used for self-removal.

A 0000001B69C	00000041B69C	0	GetStartupInfoA
<b>A</b> 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<sup>7</sup> encoding functions.

<b>A</b> 00000001B6B0	00000041B6B0	0	ABCDEFGHIJKLMNOPQRSTUVWXYZabc
A 0000001B6F8	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.

<b>A</b> 00000001B798	000000418798	0	Prsionaljrg
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 0000001BAE4	00000041BAE4	0	cfp.exe
	Fi	gure 2	1. 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	ьрррр
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
<ul> <li></li></ul>	0000004400000	<u> </u>	

Figure 22. List of common usernames and passwords.

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

<sup>&</sup>lt;sup>7</sup> http://en.wikipedia.org/wiki/Base64



#### November 2014

<b>A</b> 00000001C084	00000041C084	0	at \\%s %d:%d %s
A 00000001C098	00000041C098	0	F:\NewArea.exe
<b>A</b> 00000001C0A8	00000041CQA8	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 (OmpnbWk7
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.



November 2014

👌 🗿 🥝 🕘 💽 💌 🛨 📖				
∃-aop_unpacked.exe	pFile	Data	Description	Value
MAGE_DOS_HEADER	0000010C	014C	Machine	IMAGE_FILE_MACHINE_I386
MS-DOS Stub Program	0000010E	0004	Number of Sections	
IMAGE_NT_HEADERS	00000110		Time Date Stamp	2014/05/29 Thu 14:31:10 UTC
Signature	00000114	00000000	Pointer to Symbol Table	
IMAGE_FILE_HEADER	00000118	00000000	Number of Symbols	
IMAGE_OPTIONAL_HEADER	0000011C	00E0	Size of Optional Header	
IMAGE_SECTION_HEADER .text	0000011E	010F	Characteristics	
IMAGE_SECTION_HEADER .rdata			0001	IMAGE_FILE_RELOCS_STRIPPED
IMAGE_SECTION_HEADER .data			0002	IMAGE_FILE_EXECUTABLE_IMAGE
IMAGE_SECTION_HEADER .rsrc			0004	IMAGE_FILE_LINE_NUMS_STRIPPED
SECTION .text			0008	IMAGE_FILE_LOCAL_SYMS_STRIPPED
SECTION .rdata			0100	IMAGE_FILE_32BIT_MACHINE
IMPORT Address Table				
IMPORT Directory Table				
IMPORT DLL Names				
IMPORT Hints/Names				
SECTION .data				
SECTION .rsrc				

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.

e View Go Help					
🎐   🔾 😋 🤤 🕑 💌 💌 生   📟					
aop_unpacked.exe	pFile	Data	Description	Value	
IMAGE_DOS_HEADER	00000120	010B	Magic	IMAGE_NT_OPTIONAL_HDR32_MAG	IC
MS-DOS Stub Program	00000122	06	Major Linker Version		
E IMAGE_NT_HEADERS	00000123	00	Minor Linker Version		
Signature	00000124	00016000	Size of Code		
IMAGE_FILE_HEADER	00000128	0000A000	Size of Initialized Data		
MAGE_OPTIONAL_HEADER	0000012C	00000000	Size of Uninitialized Data		
IMAGE_SECTION_HEADER .text	00000130	000154EC	Address of Entry Point		
IMAGE_SECTION_HEADER .rdata	00000134	00001000	Base of Code		
IMAGE_SECTION_HEADER .data	00000138	00017000	Base of Data		
- IMAGE_SECTION_HEADER .rsrc	0000013C	00400000	Image Base		
- SECTION .text	00000140	00001000	Section Alignment		
.rdata ECTION .rdata	00000144	00001000	File Alignment		
SECTION .data	00000148	0004	Major O/S Version		
	0000014A	0000	Minor O/S Version		
	0000014C	0000	Major Imago Varcian		►

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<sup>8</sup>.

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

<sup>&</sup>lt;sup>8</sup> <u>http://www.on-time.com/rtos-32-docs/rttarget-32/programming-manual/compiling/borland-delphi.htm</u>



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  $\geq$  EP and EP  $\leq$  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)  $\leq$  0x154EC (EP)  $\leq$  0x16345 (.text RVA+VirtualSize).

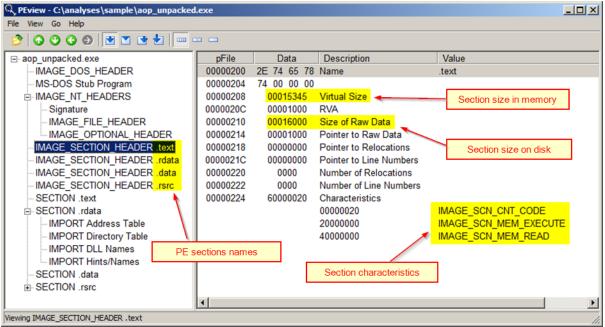


Figure 27. PE section view in PEview tool.

# 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.



November 2014

Settings ?											
	aop_unpa	cked.exe									
	Module Name	ł	Impor	ts	OFTs		TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)	
File: aop_unpacked.exe	000192A0		N/A		00018	D30	00018D34	00018D38	00018D3C	00018D40	
Dos Header Nt Headers	szAnsi		(nFun	ctions)	Dword	I	Dword	Dword	Dword	Dword	
I File Header	KERNEL32.DI	L	74		00000	000	0000000	0000000	000192A0	000170CC	
Optional Header	ADVAPI32.dl		38		00000	000	0000000	0000000	000192AD	00017000	
Data Directories [x]	AVICAP32.dl		2		00000	000	00000000	00000000	000192BA	0001709C	
ection Headers [x]	GDI32.dll		8		00000	000	00000000	00000000	000192C7	000170A8	
port Directory esource Directory	MFC42.DLL		17		00000	000	00000000	00000000	000192D1	000171F8	
ress Converter	MSVCP60.dll		10		00000	000	0000000	00000000	000192DB	00017240	
lency Walker	MSVCRT.dll		40		00000	000	0000000	00000000	000192E7	0001726C	
r				1	_	1					
	OFTs	FTs (IA	T)	Hint		Name					
dder isassembler		_									
der	Dword	Dword		Word		szAnsi					
rce Editor	N/A	000193	46	0000		lstrcpy/	1				
lity	N/A	000193	50	0000		SetEver	nt				
	N/A	000193	5A	0000		Interloc	kedExchange				

#### 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	0000000	0000000	0000000	000192A0	000170CC
ADVAPI32.dll	38	0000000	0000000	0000000	000192AD	00017000
AVICAP32.dll	2	0000000	0000000	00000000	000192BA	0001709C
GDI32.dll	8	0000000	00000000	00000000	000192C7	000170A8
MFC42.DLL	17	0000000	0000000	00000000	000192D1	000171F8
MSVCP60.dll	10	0000000	0000000	00000000	000192DB	00017240
MSVCRT.dll	40	0000000	0000000	00000000	000192E7	0001726C
MSVFW32.dll	7	0000000	0000000	00000000	000192F2	00017310
SHELL32.dll	1	0000000	0000000	00000000	000192FE	00017330
SHLWAPI.dll	1	0000000	0000000	0000000	0001930A	00017338
USER32.dll	39	0000000	0000000	00000000	00019316	00017340
WININET.dll	1	0000000	0000000	0000000	00019321	000173E0
WS2_32.dll	19	0000000	0000000	0000000	0001932D	000173E8
WTSAPI32.dll	2	0000000	0000000	0000000	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 20 Eulections imported from wtspni22 dll				

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 22. Colored functions increased from Lowe 122 dll					

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.



November 2014

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

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		Import	s	OFTs		TimeDateStamp	ForwarderC	hain:	Name RVA	FTs (IAT)
0000CB29		N/A		0000C	980	000000984	0000C988		0000C98C	000000990
szAnsi		(nFund	tions)	Dword		Dword	Dword		Dword	Dword
KERNEL32.DLL		6		00000	000	0000000	00000000		00025B1C	00025A98
ADVAPI32.dll		1		00000	000	0000000	00000000		00025B29	00025AB4
AVICAP32.dll		1		00000	000	0000000	00000000		00025B36	00025ABC
GDI32.dll		1		00000	000	0000000	00000000		00025B43	00025AC4
MFC42.DLL		1		00000	000	0000000	00000000		00025B4D	00025ACC
MSVCP60.dll		1		00000	000	0000000	00000000		00025857	00025AD4
MSVCRT.dll		1		00000	000	0000000	00000000		00025B63	00025ADC
MSVFW32.dll		1		00000	000	0000000	00000000		00025B6E	00025AE4
OFTs	FTs (IA	T)	Hint		Name					
Dword	Dword		Word		szAnsi					
N/A	000250	18	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.



November 2014

Resource Hacker - C:\analyses	\sample\aop_unpacke	
File Edit View Action Help	.0	
	Compile Script 1*9++8C2@8?BG?H1*9++8C2@8?BG?H1*9++8C2@8?BG?H1*9++8C2@	8?BC
Line: 1	150	F

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.

🔣 Resource I	lacker - C:\analyses\sample\aop_unpacked.exe	
File Edit Viev	v Action Help	
	FA@6>D5H3>*@@;B;?>6@JI Compile Script	
E Itmar Icon Icon G Urrsio	Save all resources Save [1,=/4.1FA@6>D5H3>*@@;B;?>6@JI ] resources Save [1,=/4.1FA@6>D5H3>*@@;B;?>6@JI : >,0;00@:C94HCF : 0] Replace Resource [1,=/4.1FA@6>D5H3>*@@;B;?>6@JI : >,0;00@:C94HCF : 0] Delete Resource [1,=/4.1FA@6>D5H3>*@@;B;?>6@JI : >,0;00@:C94HCF : 0] Change Language [1,=/4.1FA@6>D5H3>*@@;B;?>6@JI : >,0;00@:C94HCF : 0]	9++8C2@8?BG?H1*9++8C2@8?BG?H1*9++8C2
Line: 1	150	

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.

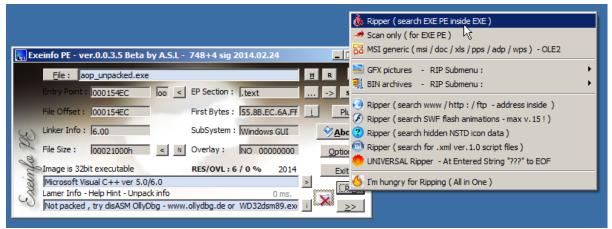


Figure 43. File ripping in Exeinfo PE.

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.

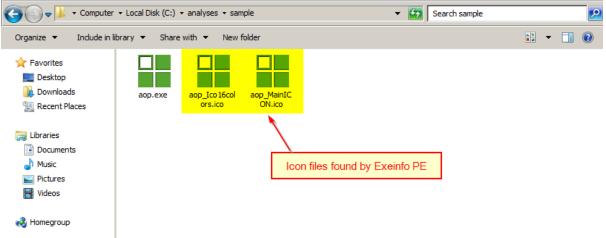


Figure 44. Icon files found by Exeinfo PE.

# 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.



November 2014

🕞 🔍 マ 🚺 C: \analyse	s\results		<ul> <li>Search results</li> </ul>		2
Organize 🔻 Include in	library ▼ Share with ▼ New folder				- 🗆 💽
🔆 Favorites	Name ^	Date modified	Туре	Size	Ň
🧮 Desktop	📔 screenshots	9/26/2014 11:46 AM	File folder		
鷆 Downloads	aop_unpacked.exe	9/26/2014 11:43 AM	Application	132 KB	
📳 Recent Places	extracted_strings.txt	9/26/2014 11:45 AM	Text Document	68 KB	
Paul -	susp_resource.txt	9/26/2014 11:45 AM	Text Document	1 KB	
Libraries Documents					
Music					
Pictures					
Videos					
🍓 Homegroup					
🖳 Computer					
👊 Network					
	Figure 45. Analysis results ir	C:\analyses\result	s directory.		

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.

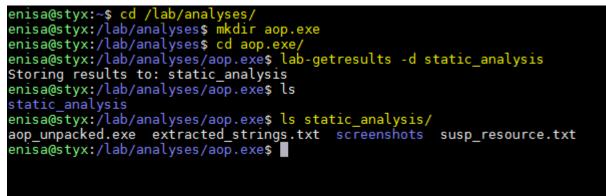


Figure 46. Downloading results to Styx.

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



- November 2014
- 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*.



November 2014

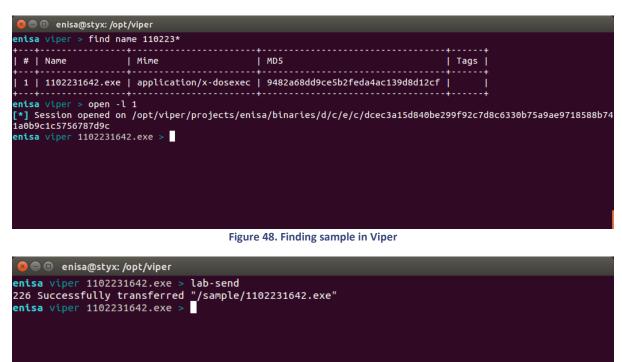


Figure 49. Sending sample to the Winbox machine.

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.



November 2014

🛃   🛃 🗉 🚍 🚳 🖌 😤	X #	1			
Process	CPU	Private Bytes	Working Set	PID Description Company Name	
System Idle Process	96.12	0 K	12 K	0	
System	0.45	44 K	760 K	4	
Interrupts	1.23	0 K	0 K	n/a Hardware Interrupts and DPCs	
smss.exe		256 K	556 K	272	
Csrss.exe	0.05	1,152 K	2,548 K	348	
🗉 🔝 wininit.exe		856 K	2,736 K	400	
services.exe		3,592 K	5,748 K	492	
svchost.exe		2,724 K	5,404 K	624 Host Process for Windows S Microsoft Corporation	
VBoxService.exe	< 0.01	1,596 K	3,492 K	680	
svchost.exe		2,416 K	4,736 K	732 Host Process for Windows S Microsoft Corporation	
svchost.exe		14,720 K	12,476 K	828 Host Process for Windows S Microsoft Corporation	
audiodg.exe		15,012 K	13,640 K	3832	
svchost.exe	< 0.01	29,764 K	32,940 K	876 Host Process for Windows S Microsoft Corporation	
dwm.exe		1,072 K	3,236 K	1564 Desktop Window Manager Microsoft Corporation	
svchost.exe		9,996 K	15,540 K	920 Host Process for Windows S Microsoft Corporation	
svchost.exe	0.07	5,784 K	10,456 K	1076 Host Process for Windows S Microsoft Corporation	
svchost.exe	0.03	7,804 K	8,564 K	1172 Host Process for Windows S Microsoft Corporation	
spoolsv.exe		4,460 K	5,872 K	1292 Spooler Sub System App Microsoft Corporation	
svchost.exe		6,564 K	4,160 K	1332 Host Process for Windows S Microsoft Corporation	
svchost.exe	0.05	4,228 K	6,492 K	1416 Host Process for Windows S Microsoft Corporation	
taskhost.exe		6,904 K	6,228 K	1608 Host Process for Windows T Microsoft Corporation	
FileZilla Server.exe	0.09	1,216 K	3,268 K	1648 FileZilla Server FileZilla Project	
svchost.exe		1,120 K	3,304 K	1512 Host Process for Windows S Microsoft Corporation	
SearchIndexer.exe		14.876 K	8.012 K	976 Microsoft Windows Search I., Microsoft Corporation	
sppsvc.exe		4.200 K	3.972 K	3480 Microsoft Software Protectio Microsoft Corporation	
sychost.exe	0.04	8.372 K	9.900 K	3588 Host Process for Windows S., Microsoft Corporation	
sass.exe		2.912 K	6.528 K	508 Local Security Authority Proc Microsoft Corporation	
sm.exe		1.276 K	2.468 K	516	
Csrss.exe	0.18	1.264 K	3.664 K	408	
winlogon.exe		1.628 K	3.404 K	448	
explorer.exe	0.17	28 808 K	44 460 K	1596 Windows Explorer Microsoft Corporation	
VBoxTray.exe	0.01	1.140 K	3.752 K	572 VirtualBox Guest Additions Tr Oracle Corporation	
FileZilla Server Interface.exe	0.02	1.620 K	6.496 K	928 FileZilla Server FileZilla Project	
Greenshot.exe	0.02	27.048 K	33.952 K	1400 Greenshot Greenshot	
	1.47	8.820 K	14.892 K	3716 Sysintemals Process Explorer Sysintemals - www.sysi	ter
No Procedu and	1.47	0,020 1	14,002 N	or to cyclicorrule i roodos Explorer oyonicel Idia "mimiayal	

Figure 50. Process Explorer window.

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

ile Edit Event Filter Tools Options	Help			
🌫 🖬   🔍 🕸 📴   🗟 🕸 -	E   👭 📕   🌋 🔜 🖉			
Time of Day Process Name	PID Operation	Path	Result	Detail
3:31:36.2823490 PM 🔳 Search odexer.exe	976 🛃 File System Control	C:	SUCCESS	Control: FSCTL_READ_USN_JOUF
3:31:36.2824525 PM 🔳 SearchIndexer.exe	976 SystemControl	C:	SUCCESS	Control: FSCTL_READ_USN_JOUF
3:31:36.2834148 FM TAUDIODG.EXE	1008 🚑 Thread Create		SUCCESS	Thread ID: 1752
3:31:36.2847332 PM 🔳 svchost.exe	624 🕂 RegCreateKey	HKLM\System\CurrentControlSet\Contr	REPARSE	Desired Access: All Access
3:31:36.2848148 PM 🔳 svchos		HKLM\System\CurrentControlSet\Contr	SUCCESS	Desired Access: All Access, Disposi
31:36.2848930 PN 📰 svchos 2	. Clear	HKLM\System\CurrentControlSet\Contr	SUCCESS	Desired Access: All Access
31:36.2849737 PM 📰 svchost.exe		HKLM\System\CurrentControlSet\Contr	SUCCESS	
:31:36.2850102 PM svchost.exe	624 RegQueryValue	HKLM\System\CurrentControlSet\Contr	NAME NOT FOU	IND Length: 44
3:31:36.2850	RegEnumKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	Index: 0, Name: ##?#HDAUDIO#F
3:31:36.2850 1. Disable event captur	e RegOpenKey	HKLM\System\CurrentControlSet\Contr	. SUCCESS	Desired Access: Read
31:36.2851303 FM = svchosl.exe	oz4 🕂 RegEnumKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	Index: 0, Name: #eAuxInTopo
3:31:36.2851619 PM 💽 svchost.exe	624 KegOpenKey	HKLM\System\CurrentControlSet\Contr	. SUCCESS	Desired Access: Read
3:31:36.2852040 PM 💽 svchost.exe	624 KegOpenKey	HKLM\System\CurrentControlSet\Contr	NAME NOT FOU	IND Desired Access: Read
:31:36.2852461 PM 💽 svchost.exe	624 RegCloseKey	HKLM\System\CurrentControlSet\Contr	. SUCCESS	
:31:36.2852774 PM 💽 svchost.exe	624 KegEnumKey	HKLM\System\CurrentControlSet\Contr	. SUCCESS	Index: 1, Name: #eAuxInWave
:31:36.2853204 PM 💽 svchost.exe	624 RegOpenKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	Desired Access: Read
:31:36.2853861 PM 💽 svchost.exe	624 KegOpenKey	HKLM\System\CurrentControlSet\Contr	. NAME NOT FOU	IND Desired Access: Read
:31:36.2854287 PM 💽 svchost.exe	624 KegCloseKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	
31:36.2854643 PM 💽 svchost.exe	624 KegEnumKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	Index: 2, Name: #eCDInTopo
:31:36.2855098 PM 💽 svchost.exe	624 KegOpenKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	Desired Access: Read
:31:36.2855717 PM 💽 svchost.exe	624 KegOpenKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	Desired Access: Read
:31:36.2856619 PM 💽 svchost.exe	624 🕂 RegQueryValue	HKLM\System\CurrentControlSet\Contr	. SUCCESS	Type: REG_DWORD, Length: 4, D
:31:36.2857079 PM svchost.exe	624 KegCloseKey	HKLM\System\CurrentControlSet\Contr	SUCCESS	
21.20 2057401 DM	<u></u>	UPUM\ 0	енестес	T DEC CZ I
				• • •

Figure 51. Disabling event capture and clearing Process Monitor

After starting Regshot check "Scan dir" option and set it to C:\.



Artifact analysis fundamentals

Artifact analysis training material

#### November 2014

義 Regshot 1.9.0 x86 Unicode	- • •
Compare logs save as:	1st shot
	2nd shot
☑ <u>S</u> can dir 1[;dir 2;dir 3;;dir nn]:	Compare
C:\	Clear
Output path:	Quit
C:\analyses\results\regshot	About
Add comment into the log:	
	English 👻

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.

8	Take Snapsh	ot of Virtua	l Machine
	Snapshot <u>N</u> ame		
	DIRTY - before	dynamic	
	Snapshot <u>D</u> esci	ription	
	Snapshot befo analysis of the	re starting dyi	
	ОК	Cancel	<u>H</u> elp

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/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	l
	nisa/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.

👞 Regshot 1.9.0 x86 Unicode 👘 🛛	
Compare logs save as:	Shot
<pre>✓ Scan dir1[;dir2;dir3;;dir nn]:</pre> C:\	Load
Output path: C:\analyses\results\regshot	Quit About
Add comment into the log:	English 🔻

Figure 54. Taking first shot in Regshot

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



🗇 Process Monitor - Sysinternals: www.sysinternals.com						
<u>F</u> ile <u>E</u> dit E <u>v</u> ent Fi <u>l</u> ter <u>T</u> ools <u>O</u> ptions <u>H</u> elp						
🗃 🖬   💸 🕅 🖾   😽 🛆 💮   [	E   # 5   <u># 5  </u>	L				
Time of Day Process Name Pl	D Operation	Path	Result			
Start capture						
٠ ( ا			۲.			
No events (capture disabled)	Backed by virtual memory					

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.

👗 Regshot 1.9.0 x86 Unicode	- • -
Compare logs save as: Plain <u>T</u> XT	1st shot
☑ <u>S</u> can dir1[;dir2;dir3;;dir nn]: C:\	Shot and Save
Output path:	Quit
C:\analyses\results\regshot	About
Add comment into the log:	English 🔻
Dirs:10603 Files:55815	Time: 18s562ms

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.

PU Usage: 7.28% Commit (	harge: 15.85%	Processes: 38	Physical Usag	e: 31.52%	
1102231642.exe	0.15	904 K	2,384 K	1272	
Regshot-x86-Unicode.ex	ce < 0.01	9,568 K	17,532 K	3732	
Tcpview.exe	0.52	6,132 K	10,772 K	1252	
Drocexp.exe	3.49	9,300 K	15,804 K	3716 Sysintemals Process Explorer Sysintemals - www.sysinter	
Greenshot.exe	< 0.01	39,112 K	53,880 K	1400 Greenshot Greenshot	
🔁 FileZilla Server Interface	.exe 0.07	1,620 K	6,496 K	928 FileZilla Server FileZilla Project	
👸 VBox Tray.exe	0.02	1,124 K	3,736 K	572 VirtualBox Guest Additions Tr Oracle Corporation	
explorer.exe	0.21	35,244 K	46,592 K	1596 Windows Explorer Microsoft Corporation	
winlogon.exe		1,628 K	3,496 K	448	
CSISS.exe	1.18	1,312 K	3,932 K	408	

Figure 57. New malware process.

Process Explorer uses a distinct colour scheme to highlight various processes<sup>9</sup>. 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).

Csrss.exe	1.62	1,312 K	3,944 K	408	
winlogon.exe		1,628 K	3,496 K	448	
explorer.exe	8.75	35,172 K	46,564 K	1596 Wind s Explorer Microsoft Corporation	
😵 VBoxTray.exe	0.02	1,124 K	3,736 K	572 VirtualBox Guest Additions Tr Oracle Corporation	
FileZilla Server Interface.exe	0.05	1,620 K	6,496 K	928 FileZilla Server FileZilla Project	
Creenshot.exe	0.01	42,700 K	57,512 K	1400 Greenshot Greenshot	
Drocexp.exe	8.49	9,424 K	16,012 K	3716 Sysintemals Process Explorer Sysintemals - www.sysinter	
Tcpview.exe	0.49	6,132 K	10,772 K	1252	
Regshot-x86-Unicode.exe		9,568 K	17,532 K	3732	
1102231642.exe	2.00	976 K	2,696 K	1272	
win32.exe		1,836 K	5,916 K	2340	
iexplarer.exe		1,036 K	3,004 K	3792	
debug.exe		1,040 K	3,036 K	3636	
sysedit.exe	0.30	1,036 K	3,004 K	888	
				-	
CPU Usage: 23.44% Commit Charge	:16.11%	Processes: 41	Physical Usag	e: 32.03%	

Figure 58. Malware process spawning child processes.

<sup>&</sup>lt;sup>9</sup> http://www.microsoft.com/security/sir/strategy/default.aspx#!malwarecleaning\_explorer



November 2014

CSrss.exe	1.20	1,312 K	3,944 K	408
winlogon.exe		1,628 K	3,496 K	448 😞
📄 🥽 explorer.exe	0.12	35,172 K	46,564 K	1596 Windows Explorer Microsoft Corporation
👸 VBoxTray.exe	0.02	1,124 K	3,736 K	572 VirtualBox Guest Additions Tr Oracle Corporation
🔁 FileZilla Server Interface.exe	0.04	1,620 K	6,496 K	928 FileZilla Server FileZilla Project
C Greenshot.exe	0.02	47,272 K	62,028 K	1400 Greenshot Greenshot
💓 procexp.exe	1.64	9,424 K	16,012 K	3716 Sysintemals Process Explorer Sysintemals - www.sysinter
Tcpview.exe	0.41	6,132 K	10,772 K	1252
Regshot-x86-Unicode.exe		9,568 K	17,532 K	3732
∎iii win32.exe		1,836 K	5,916 K	2340
iexplarer.exe		1,036 K	3,004 K	3792
debug.exe		1,040 K	3,036 K	3636
sysedit.exe		1,036 K	3,004 K	888

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).

win32.exe:23	340 Properties				- • •
TCP/IP	Security	Enviro	nment	Job	Strings
Image	Performance	e   F	erformanc	e Graph	Threads
Image File					
Version:	n/a				
Build Time	: Fri Sep 11 03:	35:02 1987	,		
Path (Ima	ige is probably pa	cked):			
C:\Users	ENISA (AppData)	\Local\Tem	p\win32.e>	ĸe	Explore
Command	line:				
C:\Users	ENISA (AppData	\Local\Tem	p\win32.e>	ĸe	
Current di	-				
	ses\sample\				
Autostart	Location:				
n/a					Explore
Parent:	1102231642.exe	(1272)		ſ	Verify
User:	ENISA-PC\ENISA	i.		L r	
Started:	2:14:57 PM 9/1	0/2014			Bring to Front
Comment:					Kill Process
VirusTotal:			<u>S</u> ul	bmit	
Data Execu	tion Prevention (I	DEP) Statu	: DEP		
Address Sp	ace Load Random	ization:	<n a=""></n>		
				<u>Ο</u> Κ	) #

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.

Security     Environment     Job     Strings       Printable strings found in the scan:	Printable strings found in the scan:	Image	Performance	Performanc	e Graph	Threads	TCP/IP
الا         ال           الا         ال           ال                     ال                     ال                     I	■ * * * * * * * * * * * * * * * * * * *	Security	En En	vironment	Job		Strings
الله الله الله الله الله الله الله		Printable strir		an:			
	ゴ Bjj ゴ Bjj Biii	iii iii iii iii iii Bijj Bijj Bijj iii					

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.



Artifact analysis fundamentals Artifact analysis training material

November 2014

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.

> persort.com http://persort.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.

Page 34



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 %IX.ttp Arial %IX.png POST Figure 65. Group of other suspicious strings.

```
btPTR

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
```

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\HideFileExt:
0x0000000
HKU\S-1-5-21-606041777-3127973734-2451401058-
1001\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt:
0x0000000
```

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-F4EAC67.pf
C:\Windows\Prefetch\WIN32.EXE-31D65D18.pf
C:\Windows\Prefetch\WINNST.EXE-66A7782D.pf
```

# 5.6 Process Monitor analysis

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



November 2014

le Edit Event Filter Tools O	ptions Help		
🗳 🖬   💸 🅦 🖾   衬 🗖	. 😌   🗉   🚧 🧦   🎊 🔜 🔍 💷 🛄		
Time of Day Process Name		Result	Detail
:38:46.5285784 PM -svchost.exe	Save To File	SUCCESS	Desired Access: Ma
38:46.5286174 PM 📰 svchost.exe	Events to save:	ft\Window SUCCESS	Desired Access: Re
38:46.5286527 PM 💽 svchost.exe	Events to save:	SUCCESS	
38:46.5286926 PM 💽 svchost.exe	All events	ft\Window SUCCESS	Query: Cached, Sul
38:46.5287201 PM 💽 svchost.exe	Events displayed using current filter	ft\Window SUCCESS	
38:46.5287386 PM 💽 svchost.exe		SUCCESS	Desired Access: Ma
38:46.5288649 PM 💽 svchost.exe	Also include profiling events	ft\Window SUCCESS	Desired Access: Re
38:46.5289289 PM 💽 svchost.exe	Highlighted events	SUCCESS	
38:46.5289491 PM 💽 svchost.exe	Format:	ft\Window SUCCESS	Query: Cached, Su
38:46.5289753 PM 💽 svchost.exe		ft\Window SUCCESS	Index: 0, Name: {00
38:46.5289989 PM 💽 svchost.exe	Native Process Monitor Format (PML)	ft\Window SUCCESS	Index: 1, Name: {0
38:46.5291201 PM 💽 svchost.exe	Comma-Separated Values (CSV)	ft\Window SUCCESS	Index: 2, Name: {0
38:46.5291755 PM 💽 svchost.exe	Extensible Markup Language (XML)	ft\Window SUCCESS	Index: 3, Name: {0
38:46.5291961 PM 💽 svchost.exe		ft\Window SUCCESS	Index: 4, Name: {0
38:46.5292154 PM 💽 svchost.exe	Include stack traces (will increase file size)	ft\Window SUCCESS	Index: 5, Name: {0
38:46.5292335 PM 💽 svchost.exe	Resolve stack symbols (will be slow)	ft\Window SUCCESS	Index: 6, Name: {02
38:46.5292519 PM 💽 svchost.exe		ft\Window SUCCESS	Index: 7, Name: {0
38:46.5292880 PM 💽 svchost.exe	Path: C:\analyses\results\ProcessMonitor.PML	ft\Window SUCCESS	Index: 8, Name: {04
38:46.5293104 PM 💽 svchost.exe		ft\Window SUCCESS	Index: 9, Name: {04
38:46.5294006 PM 💽 svchost.exe		ft\Window SUCCESS	Index: 10, Name: {(
38:46.5294891 PM 💽 svchost.exe	OK Cancel	ft\Window SUCCESS	Index: 11, Name: {(
38:46.5295359 PM 💽 svchost.exe		ft\Window SUCCESS	Index: 12, Name: {
			•

Figure 67. Saving Process Monitor results.

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

Only show processes still running at e Timelines cover displayed events only		t trace						
Process	Descri	Image Path	Life Time	Company	Ow	Command	Start Ti	Er 1
Explorer.EXE (1596)	Window	C:\Windows\Explorer.EXE		Microsoft	ENIS	C:\Windows\Explorer.EXE	9/5/201	n/a
VBoxTray.exe (572)	VirtualB	C:\Windows\System32\VBoxTray.exe		Oracle Co	ENIS.	"C:\Windows\System32\VBoxTray.exe"	9/5/201	. n/a
FileZilla Server Interface.exe (	9 FileZila	C:\Program Files\FileZilla Server\FileZilla Server Interfac		FileZilla Pr	ENIS	. "C:\Program Files\FileZilla Server\FileZilla Server Interfac.	. 9/5/201	. n/a
Greenshot.exe (1400)	Greenshot	C:\Program Files\Greenshot\Greenshot.exe		Greenshot	ENIS	. "C:\Program Files\Greenshot\Greenshot.exe"	9/5/201	. n/z
procexp.exe (3496)	Sysinter	C:\tools\Portable version\SysintemalsSuite\procexp.exe		Sysintem	ENIS	"C:\tools\Portable version\SysintemalsSuite\procexp.exe"	9/11/20	. n/z
Procmon.exe (1888)	Process	C:\tools\Portable version\SysintemalsSuite\Procmon.exe		Sysintem	ENIS	"C:\tools\Portable version\SysintemalsSuite\Procmon.ex.	. 9/11/20	. n/z
Procmon.exe (2252)	Process	C:\tools\Portable version\SysintemalsSuite\Procmon.exe		Sysintem	ENIS	. "C:\tools\Portable version\SysintemalsSuite\Procmon.ex	. 9/11/20	. n/z
A Topview.exe (3800)	TCP/U	C:\tools\Portable version\SysintemalsSuite\Tcpview.exe		Sysintem	ENIS	"C:\tools\Portable version\SysintemalsSuite\Tcpview.ex	9/11/20	. n/z
Regshot-x86-Unicode.exe (17	5 Regshot	C:\tools\Portable version\Regshot-1.9.0\Regshot-x86-U		Regshot	ENIS	"C:\tools\Portable version\Regshot-1.9.0\Regshot-x86	9/11/20	. n/a
E 1102231642.exe (108)		C:\analyses\sample\1102231642.exe			ENIS	. "C:\analyses\sample\1102231642.exe"	9/11/20	. 9/*
login.exe (1700)		C:\Users\ENISA\AppData\Local\Temp\login.exe			ENIS	. C:\Users\ENISA\AppData\Local\Temp\login.exe	9/11/20	. n/a
wininst.exe (960)		C:\Users\ENISA\AppData\Local\Temp\wininst.exe			ENIS	C:\Users\ENISA\AppData\Local\Temp\wininst.exe	9/11/20	. n/a
iexplarer.exe (1680)		C:\Users\ENISA\AppData\Local\Temp\explarer.exe			ENIS	C:\Users\ENISA\AppData\Local\Temp\explarer.exe	9/11/20	. n/a
notepad.exe (1376)		C:\Users\ENISA\AppData\Local\Temp\notepad.exe			ENIS.	. C:\Users\ENISA\AppData\Local\Temp\notepad.exe	9/11/20	. n/a
taskmgr.exe (2708)		C:\Users\ENISA\AppData\Local\Temp\taskmgr.exe			ENIS.	. C:\Users\ENISA\AppData\Local\Temp\taskmgr.exe	9/11/20	. n/a
	•		111					
m     m     m     scription: Host Process for Windows     monapy: Microsoft Corporation     ath: C: Windows/system32(svc     ommand: C: Windows/system32(svc     ser: NT AUTHORITY/WETWORK	Services host.exe host.exe -k		111					

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.



Artifact analysis training material

November 2014

ile Edit Event Filter	Tools	Options	He	lp			
i 🖓 🕅 🗎 🔰 🖉	🔶	▲ 🖗	-	🗈   🏘 🦐	💰 🗟 🛃 🔩 🖊		
Time of Day Proc	ess Name	P	ID (	Operation	Path	Result	Detail
:38:48.2445356 PM 🔳 11	02231642	exe 1	08 🙇	Process Start		SUCCESS	Parent PID: 1596, Command line: "C:\ana
:38:48.2445399 PM 🔳 11	02231642	.exe 1	08 🙇	Thread Create		SUCCESS	Thread ID: 1928
:38:48.2647323 PM 🔳 11	02231642	.exe 1	08 🧸	Load Image	C:\analyses\sample\1102231642.exe	SUCCESS	Image Base: 0x400000, Image Size: 0x31
:38:48.2670365 PM 📰 11	02231642	.exe 1	08 🧸	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x76e80000, Image Size: 0x
:38:48.2678935 PM 📰 11	02231642	.exe 1	08 星	CreateFile	C:\Windows\Prefetch\1102231642.EXE	NAME NOT FOUND	Desired Access: Generic Read, Dispositio
:38:48.2681388 PM 📰 11	02231642	.exe 1	08 星	CreateFile	C:\analyses\sample	SUCCESS	Desired Access: Execute/Traverse, Sync
:38:48.2687351 PM 🔳 11	02231642	.exe 1	08 🗖	Load Image	C:\Windows\System32\kernel32.dll	SUCCESS	Image Base: 0x76a70000, Image Size: 0x
:38:48.2756470 PM 🔳 11	02231642	.exe 1	08 🙇	Load Image	C:\Windows\System32\KernelBase.dll	SUCCESS	Image Base: 0x75250000, Image Size: 0x
:38:48.2764430 PM 🔳 11	02231642	lexe 1	08 🙍	RegOpenKey	HKLM\System\CurrentControlSet\Control	REPARSE	Desired Access: Read
:38:48.2764671 PM 🔳 11	02231642	.exe 1	08 🚮	RegOpenKey	HKLM\System\CurrentControlSet\Control	SUCCESS	Desired Access: Read
:38:48.2764873 PM 🔳 11	02231642	lexe 1	08 🙍	RegQueryValue	HKLM\System\CurrentControlSet\Control	NAME NOT FOUND	Length: 548
:38:48.2764989 PM 🔳 11	02231642	.exe 1	08 🚮		HKLM\System\CurrentControlSet\Control	SUCCESS	Type: REG_DWORD, Length: 4, Data: 0
:38:48.2765092 PM 🔳 11	02231642	lexe 1	08 🙍	RegCloseKey	HKLM\System\CurrentControlSet\Control		
:38:48.2765307 PM 🔳 11	02231642	.exe 1	08 🔮	RegOpenKey	HKLM\System\CurrentControlSet\Control	REPARSE	Desired Access: Query Value, Set Value
:38:48.2765440 PM 🔳 11				RegOpenKey	HKLM\System\CurrentControlSet\Control		
:38:48.2771269 PM 🔳 11	02231642	.exe 1	08 🙍	RegOpenKey	HKLM\System\CurrentControlSet\Control	REPARSE	Desired Access: Read
:38:48.2771475 PM 🔳 11				RegOpenKey	HKLM\System\CurrentControlSet\Control		
:38:48.2771613 PM 🔳 11	02231642			RegOpenKey	HKLM\Software\Policies\Microsoft\Wind		Desired Access: Query Value
:38:48.2774010 PM 🔳 11				RegQueryValue	HKLM\SOFTWARE\Policies\Microsoft\		Length: 80
:38:48.2774113 PM 🔳 11	02231642			RegCloseKey	HKLM\SOFTWARE\Policies\Microsoft\		
:38:48.2774366 PM 🔳 11				RegOpenKey	HKCU\Software\Policies\Microsoft\Wind		
:38:48.2782400 PM 🔳 11				RegOpenKey	HKLM\System\CurrentControlSet\Control		Desired Access: Query Value
:38:48.2782567 PM 🔳 11				RegOpenKey	HKLM\System\CurrentControlSet\Control		Desired Access: Query Value
:38:48.2782735 PM 🔳 11				RegQueryValue	HKLM\System\CurrentControlSet\Control		
:38:48.2785802 PM 🔳 11				Load Image	C:\Windows\System32\user32.dll	SUCCESS	Image Base: 0x765d0000, Image Size: 0x
:38:48.2788418 PM 🔳 11				Load Image	C:\Windows\System32\gdi32.dll	SUCCESS	Image Base: 0x766b0000, Image Size: 0x
:38:48.2791081 PM 🖃 11				Load Image	C:\Windows\System32\pk.dll	SUCCESS	Image Base: 0x766a0000, Image Size: 0x
:38:48.2795888 PM 🔳 11				Load Image	C:\Windows\System32\usp10.dll	SUCCESS	Image Base: 0x76900000, Image Size: 0x
:38:48.2797740 PM 🖃 11	02231642	lexe 1	08 🗸	Load Image	C:\Windows\System32\msvcrt.dll	SUCCESS	Image Base: 0x75600000, Image Size: 0x
			-	•••			h

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>'.

Process Monitor Hig	hlighting		<b>—</b> ———————————————————————————————————
Highlight entries matching Architecture	these conditions:		✓ then Indude ✓       Add     Remove
Column           Image: Column <th>Relation is is is</th> <th>Value Process Create WriteFile Process Start</th> <th>Action Include Include Include</th>	Relation is is is	Value Process Create WriteFile Process Start	Action Include Include Include
Make Filter			ancel Apply

Figure 70. Adding highlight in Process Monitor.

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



Artifact analysis training material

November 2014

ile Edit Event Filter Tool	ls Options H	lelp			
🎽 🖬   💸 🕅 🖾   🤸	🔁 🗛 🏺	🖻   🚧 📕   [			
Time of Day Process Nar	me PID	Operation	Path	Result	Detail
:38:52.4111646 PM 🔳 11022316	642.exe 108		C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Desired Access: Write At
:38:52.4112857 PM 🔳 11022316	42.exe 108	SetBasicInformation	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Creation Time: 1/1/1601
:38:52.4115855 PM 🔳 11022316			C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	
:38:52.4122974 PM 📧 11022316	42.exe 108	CreateFile	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Desired Access: Read At
:38:52.4123966 PM 🔳 11022316	42.exe 108	🛃 Query Basic Informati	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Creation Time: 9/11/2014
:38:52.4124348 PM 📧 11022316		CloseFile	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	
:38:52.4129800 PM 🔳 11022316		CreateFile	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Desired Access: Read At
:38:52.4130727 PM 🔳 11022316			C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Creation Time: 9/11/2014
:38:52.4131045 PM 🔳 11022316		CloseFile	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	
:38:52.4134628 PM 🔳 11022316		CreateFile	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Desired Access: Read D
:38:52.4137730 PM 🔳 11022316		🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\login.exe		Offset: 0, Length: 24,576
:38:52.4158487 PM 🔳 11022316			.C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	EndOfFile: 20,788
:38:52.4158989 PM 🔳 11022316		CreateFileMapping	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	SyncType: SyncTypeOth
:38:52.4159831 PM 🔳 11022316		CreateFileMapping	C:\Users\ENISA\AppData\Local\Temp\login.exe	FILE LOCKED WI	SyncType: SyncTypeCre
:38:52.4159990 PM 🔳 11022316			C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	AllocationSize: 24,576, E
:38:52.4160923 PM 🔳 11022316		CreateFileMapping	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	SyncType: SyncTypeOth
:38:52.4161524 PM 🔳 11022316		式 RegOpen Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersi		Desired Access: Query V
:38:52.4162035 PM 💽 11022316		QuerySecurityFile	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Information: Label
:38:52.4164368 PM 🔳 11022316			. C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	Name: \Users\ENISA\Ap
:38:52.4165382 PM 🔳 11022316		🌄 Process Create	C:\Users\ENISA\AppData\Local\Temp\login.exe	SUCCESS	PID: 1700, Command line
:38:52.4165425 PM 📰 login.exe		🚑 Process Start		SUCCESS	Parent PID: 108, Comma
:38:52.4165472 PM 🔳 loain.exe	1700	🔄 Thread Create		SUCCESS	Thread ID: 1100
					+

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...).

isplay entries matching th Operation 🔹 is		rocess Create		✓ then Include
Reset				Add <u>R</u> emove
Column	Relation	Value	Action	
🔽 🙆 PID	is	108	Include	
🔽 🧑 PID	is	1700	Include	
🔽 🧑 PID	is	960	Include	
🔽 🙆 PID	is	1680	Include	l
🔽 📀 PID	is	1376	Include	
🔽 📀 PID	is	2708	Include	
🔽 📀 Operation	is	RegSetValue	Include	
🔽 📀 Operation	is	WriteFile	Include	
🔽 📀 Operation	is	Process Create	Include	
🔽 这 Process Name	is	Procmon.exe	Exclude	
🔽 🐼 Process Name	is	Procexp.exe	Exclude	

Figure 72. Include filters in Process Monitor.

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



Artifact analysis training material

November 2014

ile Edit Event Filter Tools Opti	ons Help	
🛎 🖬   💸 🖗 🖾   衬 🔺 (	🖗   🖻   🛤 🐬	
Time of Day Process Name	PID Operation	Path
:38:49.2751238 PM 🔳 1102231642.exe	108 戱 RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\UserID
:38:52.4078851 PM E 1102231642.exe	108 🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\login.exe
:38:52.4100756 PM 💽 1102231642.exe	108 🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\login.exe
38:52.4137730 PM E 1102231642.exe	108 🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\login.exe
:38:52.4165382 PM 💽 1102231642.exe	108 🖉 Process Create	C:\Users\ENISA\AppData\Local\Temp\login.exe
:38:52.9648913 PM 💽 1102231642.exe	108 🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\wininst.exe
:38:52.9657230 PM 🔳 1102231642.exe	108 🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\wininst.exe
:38:52.9679465 PM 💽 1102231642.exe	108 🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\wininst.exe
:38:52.9708921 PM 💽 1102231642.exe	108 🖉 Process Create	C:\Users\ENISA\AppData\Local\Temp\wininst.exe
38:53.2297826 PM 🔳 login.exe	1700 🌋 RegSetValue	HKCU\Software\Microsoft\Internet Explorer\New Windows\PopupMgr
:38:53.2452290 PM 🔳 login.exe	1700 🌋 RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\hsfio38fiosfh398rfisjhkdsfd
:38:53.2470104 PM 🔳 login.exe	1700 🌋 RegSet Value	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden
:38:53.2472175 PM 🔳 login.exe	1700 🌋 RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt
:38:53.2472647 PM 🔳 login.exe	1700 🌋 RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\SuperHidden
:38:53.2478481 PM 🔳 login.exe	1700 🛃 WriteFile	C:\Users\ENISA\AppData\Local\Temp\skaioejiesfjoee.tmp
:38:53.3736327 PM 🔳 login.exe	1700 🌋 RegSetValue	HKLM\SOFTWARE\Microsoft\Tracing\login_RASAPI32\EnableFileTracing
:38:53.3738835 PM 🔳 login.exe	1700 🌋 RegSetValue	HKLM\SOFTWARE\Microsoft\Tracing\login_RASAP132\EnableConsoleTracing
:38:53.3742951 PM 🔳 login.exe	1700 🌋 RegSetValue	HKLM\SOFTWARE\Microsoft\Tracing\login_RASAP132\FileTracingMask
:38:53.3744639 PM 🔳 login.exe	1700 戱 RegSetValue	HKLM\SOFTWARE\Microsoft\Tracing\login_RASAP132\ConsoleTracingMask
		• • • • •

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.



November 2014

Frame	Module	Location	Address	Path
K 0	fltmgr.sys	FltRequestOperationStatusCallbac	0x87f7faeb	C:\Windows\system32\drivers\fltmgr.sy
K 1	fltmgr.sys	FtGetInpName + 0xc5c	0x87f829f0	C:\Windows\system32\drivers\fltmgr.sy
K 2	fltmgr.sys	FtGetIrpName + 0x116d	0x87f82f01	C:\Windows\system32\drivers\fltmgr.sy
K 3	fltmgr.sys	FtGetIrpName + 0x1626	0x87f833ba	C:\Windows\system32\drivers\fltmgr.sy
K 4	ntoskml.exe	lofCallDriver + 0x64	0x8287bf44	C:\Windows\system32\ntoskml.exe
K 5	ntoskml.exe	NtQueryInformationThread + 0x5cd8	0x82a51287	C:\Windows\system32\ntoskml.exe
K 6	ntoskml.exe	NtWriteFile + 0x6ee	0x82a66f09	C:\Windows\system32\ntoskml.exe
K 7	ntoskml.exe	ZwYieldExecution + 0xb5a	0x8288279a	C:\Windows\system32\ntoskml.exe
U 8	ntdll.dll	NtWriteFile + 0xc	0x76ec5ebc	C:\Windows\System32\ntdll.dll
U 9	KemelBase.dll	MapGenericMask + 0x47	0x752590e3	C:\Windows\System32\KemelBase.dll
U 10	kemel32.dll	SetFileInformationByHandle + 0xd68	0x76aaf70d	C:\Windows\System32\kernel32.dll
U 11	kemel32.dll	SetFileInformationByHandle + 0x6c7	0x76aaf06c	C:\Windows\System32\kernel32.dll
U 12	kemel32.dll	CopyFileExW + 0x4a	0x76ab0805	C:\Windows\System32\kemel32.dll
U 13	kemel32.dll	CopyFileA + 0x4d	0x76ad7d49	C:\Windows\System32\kernel32.dll
<b>U</b> 14	1102231642.exe	1102231642.exe + 0x4d70	0x404d70	C:\analyses\sample\1102231642.exe
			erties) Se	earch Source Save

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.

Path	Writers	Readers
C:\Users\ENISA\AppData\Local\Temp\iexplarer.exe	1102231642.exe	1102231642.exe, iexplarer.exe
C:\Users\ENISA\AppData\Local\Temp\login.exe	1102231642.exe	1102231642.exe, login.exe
:\Users\ENISA\AppData\Local\Temp\notepad.exe	1102231642.exe	1102231642.exe, notepad.exe
C:\Users\ENISA\AppData\Local\Temp\skaioejiesfjoee.tmp	login.exe	iexplarer.exe, notepad.exe, taskmgr.exe, wininst.exe
C:\Users\ENISA\AppData\Local\Temp\taskmgr.exe	1102231642.exe	1102231642.exe, taskmgr.exe
:\Users\ENISA\AppData\Local\Temp\wininst.exe	1102231642.exe	1102231642.exe, wininst.exe
HKCU\Software\Microsoft\Internet Explorer\New Windows\PopupMgr	iexplarer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Hidden	iexplarer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\HideFileExt	iexplarer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\SuperHidden HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\UserID	iexplarer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe 1102231642.exe	1102231642.exe, iexplarer.exe, login.exe, notepad.e
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\hsfio38fiosfh398fisjhkdsfd	iexplarer.exe, login.exe, notepad.exe, taskmgr.exe, wininst.exe	

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.

GMER 2.1.19357 WINDOWS 6.1.760		
Rootkit/Malware >>>		
Type Name	Value	✓ System         ✓ Sections         ✓ IAT/EAT         ✓ Devices         ✓ Trace I/D         ✓ Modules         ✓ Processes         ✓ Threads         ✓ Libraries         ✓ Services         ✓ Registry         ✓ Files         ✓ Quick scan         ○ C.\         ✓ ADS
		Show all 3rd party Scan <u>C</u> opy Save
GMER 2.1.19357 WINDOWS 6.1.7600	l	

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.



November 2014

Туре	Name	Value	🔽 System
text text ?	ntkinipa.exelZwSaveKeyEx + 13AD ntkinipa.exelZwSaveKeyEx + 13AD tikinipa.exelXibispatchInterrupt + 5A2 C:\Windows\system32\Drivers\PROCMDN23.SYS C:\Users\ENISA\AppData\Local\Temp\debug.exe[3636]C:\Users\ C:\Users\ENISA\AppData\Local\Temp\debug.exe[3636]C:\Users\	, 82661579 1 Byte [06] 82685F52 19 Bytes [E0, 0F, BA, F0, 07, 73, 09,] {LOOPNZ 0 The system cannot find the file specified ! entry point in "" section [0x00430E36]	✓ Sections     ✓ Sections     ✓ IAT/EAT     ✓ Devices     ✓ Trace I/O     ✓ Modules     ✓ Processes     ✓ Threads     ✓ Libraries     ✓ Services     ✓ Registry     ✓ Files     ✓ Quick sca       C:\     ✓ ADS     Show all     ③ 3rd party     Scan     Copy

#### 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

November 2014

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.

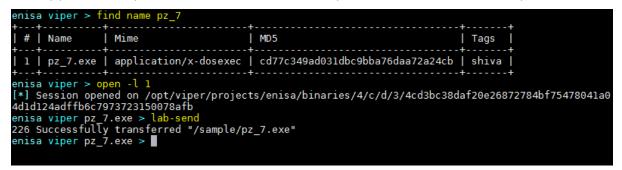


Figure 79. Sending sample to the analysis.

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 labcleanlogs 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-
<pre>12:24:04.425421 IP 10.0.0.1.80 &gt; 10.0.0.2.49242: Flags [P.], seq 1:409, ack 290, win 237, length 408 12:24:04.426014 IP 10.0.0.2.49242 &gt; 10.0.0.1.80: Flags [F.], seq 290, ack 409, win 16323, length 0 12:24:04.426680 IP 10.0.0.2.49242 &gt; 10.0.0.1.80: Flags [F.], seq 409, ack 291, win 237, length 0 12:31:45.422982 IP 10.0.0.2.49242 &gt; 10.0.0.1.80: Flags [.], ack 410, win 16323, length 0 12:31:45.423042 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [S], seq 1094090034, win 8192, options [mss 1460,nop, wscale 2,nop,nop,sackOK], length 0 12:31:45.423042 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [S.], seq 3000101690, ack 1094090035, win 29200, options [mss 1460,nop,nop,sackOK,nop,wscale 7], length 0 12:31:45.423356 IP 10.0.0.2.49243 &gt; 10.0.0.1.80: Flags [.], ack 1, win 16425, length 0 12:31:45.423853 IP 10.0.0.2.49243 &gt; 10.0.0.1.80: Flags [.], ack 413, win 237, length 412 12:31:45.423853 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [.], ack 413, win 237, length 412 12:31:45.423853 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [.], ack 413, win 237, length 0 12:31:45.423853 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [.], seq 1:409, ack 413, win 237, length 408 12:31:45.808125 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [F.], seq 413, ack 409, win 16323, length 0 12:31:45.808125 IP 10.0.0.1.80 &gt; 10.0.0.1.80: Flags [F.], seq 409, ack 414, win 237, length 0 12:31:45.808923 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [F.], seq 409, ack 414, win 237, length 0 12:31:45.808923 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [.], ack 410, win 16323, length 0 12:31:45.809916 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [.], ack 410, win 16323, length 0 12:31:45.809916 IP 10.0.0.1.80 &gt; 10.0.0.2.49243: Flags [.], ack 410, win 16323, length 0 12:31:45.809916 IP 10.0.0.2.49243 &gt; 10.0.0.1.80: Flags [.], ack 410, win 16323, length 0 12:31:45.809916 IP 10.0.0.2.49243 &gt; 10.0.0.1.80: Flags [.], ack 410, win 16323, length 0 12:31:45.809916 IP 10.0.0.2.49243 &gt; 10.0.0.1.80: Flags [.], ack 410, win 16323, length 0 12:31:45.809916 IP 10.0.0.2.49243 &gt; 10.0.0.1.80: Flags [.], ack 410, win 1</pre>

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.

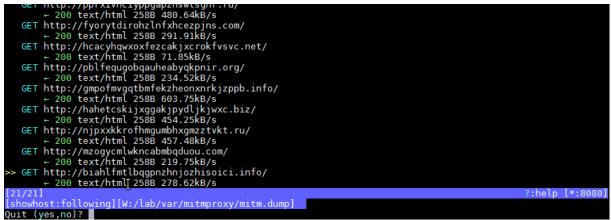


Figure 82. Quiting mitmproxy.

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

Stopping <i>tcpdump</i> 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 <i>tcpdump</i> packet capture
<pre>\$ mkdir -p /lab/analyses/pz_7.exe</pre>
<pre>\$ cd /lab/analyses/pz_7.exe</pre>
\$ 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.



November 2014

enisa@styx:/lab\$ cd analyses/pz_7.exe/net_results/pc	ans/
enisa@styx:/lab/analyses/pz 7.exe/net results/pcaps\$	lab condfile not 140010164201 ncan
net_140919164301.pcap:	153.51 kB 323.17 kB/s
<pre>enisa@styx:/lab/analyses/pz_7.exe/net_results/pcaps\$</pre>	

#### Figure 83. Sending pcap to Winbox machine.

Open the uploaded file in Wireshark on Winbox.

72       104.4037/10.0.0.1       10.0.0.2       100       10	📕 net	t_140919164301.pcap [Wireshark 1.10.8 (v1.10.8-2-g52a5244	from master-1.10)]			
Filter       Expression Clear Apply Save         0:       Time       Source       Destination         7:164.4657410.0.0.1       10.0.0.1       TCP       60 49232 > http [ACK] Seq=0 ACK=1         73164.4657110.0.0.2       10.0.0.1       TCP       60 49232 > http [ACK] Seq=1 Ack=272 wi         74164.46606910.0.0.2       10.0.0.1       HTTP 325 GET / HTTP/1.1         75164.466069010.0.0.1       10.0.0.2       TCP       54 http > 49232 [ACK] Seq=1 Ack=272 wi         76164.78445510.0.0.1       10.0.0.2       TCP       54 http > 49232 [ACK] Seq=272 Ack         78164.78520210.0.0.1       10.0.0.1       TCP       60 49232 > http [FIN, ACK] Seq=272 Ack         78164.78520210.0.0.1       10.0.0.1       TCP       60 49232 > http [ACK] Seq=273 Ack+410         80166.28780710.0.0.2       8.8.8       DNS       91 standard query ox8b0e A 2hdxoirxzp         81166.314526 8.8.8       10.0.0.2       DNS       107 standard query ox8b0e A 2hdxoirxzp         82 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [SYN] Seq=0 win=8192 L         4       m       m	<u>F</u> ile	Edit View Go Capture Analyze Statistics Telepho	n <u>y T</u> ools <u>I</u> nternals <u>H</u> elp			
Time       Source       Destination       Protocol       Length       Info         72       164.4657110.0.0.1       10.0.0.2       1CP       b0 fhttp > 49232       [SYN, Ack] Seq=0 Ack=1         73       164.4650110.0.0.2       10.0.0.1       TCP       60.49232       http [Ack] Seq=1 Ack=1       wine         74       164.465010.0.0.1       10.0.0.2       TCP       54       http > 49232       [Ack] Seq=1 Ack=272       wine         75       164.46609010.0.0.1       10.0.0.2       TCP       54       http > 49232       [Ack] Seq=272 Ack         76       164.7845510.0.0.1       10.0.0.1       TCP       60.49232       http > 49232       [FIN, Ack] Seq=272 Ack         78       164.785294       10.0.0.2       10.0.0.1       TCP       60.49232       http > 49232       [FIN, Ack] Seq=273 Ack=410         80       166.287807       10.0.0.2       8.8.8       DNS       91       standard query 0x8b0e A zhdxoirxzp         81       166.314826 8.8.8       10.0.0.2       DNS       91       standard query 0x8b0e A zhdxoirxzp         82       166.316157       10.0.0.2       0.0.0.1       TCP       66       49233       http [SVN] Seq=0 win=8192         4	0	ې 🔶 🔶 🍋 🔝 🔚 🕲 🖉 🔍	• <b>7 ⊻</b>   <b>8 1 1 1 1 1 1 1 1 1 1</b>	Q 🖭   👹	M 🍢 💥 🛛	
72       104.405/110.0.0.1       10.0.0.2       10.0.0.1       TCP       60       49232 > http [AcK] seq=0 AcK=1         73       164.4660910.0.0.2       10.0.0.1       TCP       60       49232 > http [AcK] seq=1 Ack=1 wine         74       164.4660910.0.0.1       10.0.0.1       HTTP       325 GET / HTTP/1.1         75       164.78445510.0.0.1       10.0.0.2       TCP       54 http > 49232 [AcK] seq=1 Ack=272 wind         76       164.78445510.0.0.1       10.0.0.2       TCP       60       49232 > http [FIN, AcK] seq=272 Ack         79       164.7845510.0.0.1       10.0.0.2       TCP       60       49232 > http [AcK] seq=473 Ack         79       164.78559010.0.0.2       10.0.0.1       TCP       60       49232 > http [AcK] seq=273 Ack         80       166.28780710.0.0.2       10.0.0.1       TCP       60       49232 > http [AcK] seq=473 Ack         81       166.31615710.0.0.2       8.8.8.8       DNS       91       standard query 0x8b0e A 1dxoirxzp         81       166.31615710.0.0.2       10.0.0.1       TCP       66       49233 > http [SvN] Seq=0 win=8192 L         92       Frame 63: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)       92       92       92         92       Erame 63: 208 bytes on wire (1064 bits)	Filter:		<ul> <li>Expression Clear</li> </ul>	Apply Save		
73 164.465711 10.0.0.2       10.0.0.1       TCP       60 49232 > http [AcK] seq=1 Ack=1 wine         74 164.466091 0.0.0.2       10.0.0.1       HTTP       325 GET / HTTP/1.1         75 164.466091 0.0.0.1       10.0.0.2       TCP       54 http > 49232 [AcK] seq=1 Ack=272 wi         76 164.784455 10.0.0.1       10.0.0.2       HTTP       462 HTTP/1.1       200 oK (text/htm])         77 164.784948 10.0.0.2       10.0.0.1       TCP       60 49232 > http [FIN, AcK] seq=409 Ack         79 164.785594 10.0.0.2       10.0.0.1       TCP       60 49232 > http [AcK] seq=273 Ack=410         80 166.287807 10.0.0.2       8.8.8       DNS       91 Standard query response 0x8b0e A 1         81 166.314826 8.8.8       10.0.0.1       TCP       66 49233 > http [SYN] seq=0 win=8192 L         4       III       IIII       IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII	No.				Length Info	
74 164.466069 10.0.0.2       10.0.0.1       HTTP       325 GET / HTTP/1.1         75 164.466090 10.0.0.1       10.0.0.2       TCP       54 http > 49232 [ACK] seq=1 ACk=272 wi         76 164.784455 10.0.0.1       10.0.0.2       HTTP       462 HTTP/1.1       200 ov (text/html)         77 164.784495 10.0.0.2       10.0.0.1       TCP       54 http > 49232 [ACK] seq=1 ACk=272 wi         77 164.784945 10.0.0.2       10.0.0.1       TCP       60 49232 > http [FIN, ACK] seq=272 Ack         78 164.78559 10.0.0.2       10.0.0.1       TCP       60 49232 > http [ACK] seq=273 Ack=410         80 166.287807 10.0.0.2       10.0.0.1       TCP       60 49232 > http [ACK] seq=273 Ack=410         80 166.287807 10.0.0.2       8.8.8.8       DNS       91 standard query 0x8b0e A zhdxoirxzp         81 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [SVN] seq=0 win=8192 L          Trame 63: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)       95 246.170.150 (95.246.170.150)         © Ethernet II, Src: Cadmusco_66:25:2b (08:00:27:66:25:2b), Dst: Cadmusco_a7:7e:0e (08:00:27:a7:7e:0e)       91         © Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 95.246.170.150 (95.246.170.150)       92         © User Datagram Protocol, Src Port: street-stream (1736), Dst Port: funk-license (1787)       92         © Data (166 bytes)       9						
75 164.466090 10.0.0.1       10.0.0.2       TCP       54 http > 49232 [AcK] Seq=1 Ack=272 wi         76 164.784455 10.0.0.1       10.0.0.2       HTTP       462 HTTP/1.1 200 0K (text/html)         77 164.78502 10.0.0.2       10.0.0.1       TCP       60 49232 > http [FN, AcK] Seq=272 Ack         78 164.785202 10.0.0.1       10.0.0.2       TCP       54 http > 49232 [FN, AcK] Seq=272 Ack         79 164.785594 10.0.0.2       10.0.0.1       TCP       60 49232 > http [AcK] Seq=273 Ack=410         80 166.287807 10.0.0.2       8.8.8.8       DNS       91 standard query 0x8b0e A 2 hdxoirxzp         81 166.314826 8.8.8.8       10.0.0.2       DNS       107 standard query response 0x8b0e A 1         82 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [SYN] Seq=0 win=8192 L         Image: Transe 63: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)       Image: Transe 63: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)         Image: Transe 63: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)       Image: Transe 63: 208 bytes on wire (1664 bits), 208 bytes captured (1664 bits)         Image: Transe 63: 208 bytes on wire (1664 bits), 208 over sindard query over sindard query over sind 4, Src: 10.0.0.2 (10.0.0.2), Dst: 95.246.170.150 (95.246.170.150)         Image: Transe 63: 208 bytes on wire (1664 bits), 208 00 00 00 25 ff 6      ( !u						Seq=1 ACK=1 WITH=
76       164.784455       10.0.0.1       10.0.0.2       HTTP       462       HTTP/1.1       200       0K       (text/html)         77       164.784948       10.0.0.2       10.0.0.1       TCP       60       49232 > http       [FIN, ACK] seq=272       Ack         78       164.785202       10.0.0.1       10.0.0.2       TCP       54       http > 49232       [FIN, ACK] seq=273       Ack       4490       Ack         79       164.785202       10.0.0.2       10.0.0.1       TCP       54       http > 49232       [FIN, ACK] seq=273       Ack       4400       Ack       seq=273       Ack       4400       Ack       Seq=273       Ack       4410       Ack						Con 1 Ack 272 Md
77 164.784948 10.0.0.2       10.0.0.1       TCP       60 49232 > http [FIN, AcK] seq=272 AcK         78 164.785202 10.0.0.1       10.0.0.2       TCP       54 http > 49232 [FIN, AcK] seq=409 AcK         79 164.785594 10.0.0.2       10.0.0.1       TCP       60 49232 > http [AcK] seq=409 AcK         80 166.287807 10.0.0.2       8.8.8.8       DNS       91 standard query 0x8b0e A zhdxoirxzp         81 166.314826 8.8.8.8       10.0.0.2       DNS       107 standard query response 0x8b0e A 1         82 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [SYN] seq=0 win=8192 L         Image: Standard query versions 0x8b0e A 1       10.0.0.2       DNS       107 standard query response 0x8b0e A 1         82 166.316157 10.0.0.2       10.0.0.2       DNS       107 standard query response 0x8b0e A 1         Image: Standard query versions 0x8b0e A 1       10.0.0.2       DNS       107 standard query response 0x8b0e A 1         Image: Standard query versions 0x8b0e A 1       10.0.0.2       DNS       107 standard query versionse 0x8b0e A 1         Image: Standard query versions 0x8b0e A 1       10.0.0.2       DNS       107 standard query versionse 0x8b0e A 1         Image: Standard query versions 0x8b0e A 1       10.0.0.2       10.0.0.1       TCP       66 49233 > http [SYN] seq=0 win=8192 L         Image: Standard query versions 0x8b0e A 20						
78       164.785202 10.0.0.1       10.0.0.2       TCP       54       http > 49232       [FIN, ACK] seq=409 Ack         79       164.785594 10.0.0.2       10.0.0.1       TCP       60       49232 > http [AcK] seq=273 Ack=410         80       166.287807 10.0.0.2       8.8.8.8       DNS       91       standard query 0x8b0e A zhdxoirxzp         81       166.314826 8.8.8.8       10.0.0.2       DNS       107 standard query response 0x8b0e A 1         82       166.316157 10.0.0.2       10.0.0.1       TCP       66       49233 > http [SYN] Seq=0 win=8192 L         Image: the control of the control						
79 164.785594 10.0.0.2       10.0.0.1       TCP       60 49232 > http [ACK] seq=273 Ack=410         80 166.287807 10.0.0.2       8.8.8.8       DNS       91 Standard query 0x8b0e A zhdxoirxzp         81 166.314826 8.8.8       10.0.0.2       DNS       107 Standard query response 0x8b0e A 1         82 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [SYN] Seq=0 Win=8192 L         Image: Standard query response 0x8b0e A 1       10.0.0.1       TCP       66 49233 > http [SYN] Seq=0 Win=8192 L         Image: Standard query response 0x8b0e A 1       10.0.0.1       TCP       66 49233 > http [SYN] Seq=0 Win=8192 L         Image: Standard query response 0x8b0e A 1       10.0.0.1       TCP       66 49233 > http [SYN] Seq=0 Win=8192 L         Image: Standard query response 0x8b0e A 1       10.0.0.1       TCP       66 49233 > http [SYN] Seq=0 Win=8192 L         Image: Standard query response 0x8b0e A 1       10.0.0.2       Dott       TCP       66 49233 > http [SYN] Seq=0 Win=8192 L         Image: Standard query response 0x8b0e A 1       10.0.0.2       Dott       TCP       66 49233 > http [SYN] Seq=0 Win=8192 L         Image: Standard query response 0x8b0e A 1       10.0.0.2       Dott       Internet Trip Standard query response 0x8b0e A 2         Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 95.246.170.150 (95.246.170.150)       Internet Trip Standard query respons						
80 166.287807 10.0.0.2       8.8.8.8       DNS       91 standard query 0x8b0e       A zhdxoirxzp         81 166.314826 8.8.8.8       10.0.0.2       DNS       107 Standard query response 0x8b0e       A 1         82 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [svN] seq=0 win=8192 L         Image: Constraint of the sequence of t						
81 166.314826 8.8.8.8       10.0.0.2       DNS       107 standard query response 0x8b0e A 1         82 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [SYN] seq=0 win=8192 L         Image: Constraint of the standard of the sta						
82 166.316157 10.0.0.2       10.0.0.1       TCP       66 49233 > http [SYN] seq=0 win=8192 L						
<pre></pre>						
<pre>General State State</pre>	4		10.0.0.1	ICP	00 49255 > http [51N]	Seq=0 will=0192 L
<pre> B Ethernet II, Src: Cadmusco_66:25:2b (08:00:27:66:25:2b), Dst: Cadmusco_a7:7e:0e (08:00:27:a7:7e:0e) D therenet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 95.246.170.150 (95.246.170.150) D User Datagram Protocol, Src Port: street-stream (1736), Dst Port: funk-license (1787) D Data (166 bytes)  0000 08 00 27 a7 7e 0e 08 00 27 66 25 2b 08 00 45 00</pre>						r
<pre> Internet Protocol version 4, src: 10.0.0.2 (10.0.0.2), bst: 95.246.170.150 (95.246.170.150) User Datagram Protocol, Src Port: street-stream (1736), Dst Port: funk-license (1787) Data (166 bytes)  0000 08 00 27 a7 7e 0e 08 00 27 66 25 2b 08 00 45 00'.~ 'f%+E. 0010 00 c2 04 28 00 00 80 11 21 75 0a 00 00 25 ff 6(!u 0020 aa 96 06 c8 06 fb 00 ae dc 8b fd b8 83 37 91 59</pre>						
<pre> 3 User Datagram Protocol, Src Port: street-stream (1736), Dst Port: funk-license (1787) 3 Data (166 bytes) 3 Data (166 b</pre>						
Data (166 bytes)						
0000       08       00 27       a7       7e       0e       08       00       27       66       25       2b       08       00       45       00      ,, 'f%+E.         0010       00       c2       24       28       00       08       01       21       75       0a       00       00       25       f      ,, 'fw+E.         0020       aa       96       6c       80       fb       0a       ac       8b       fd       b8       33       7       91       59      ,,			tream (1736), Dst Port: f	unk-licens	e (1787)	
0010       00 c2 04 28 00 00 80 11 21 75 0a 00 00 02 5f f6      (!u	+ Da	ta (166 bytes)				
0010       00 c2 04 28 00 00 80 11 21 75 0a 00 00 02 5f f6      (!u						
0010       00 c2 04 28 00 00 80 11 21 75 0a 00 00 02 5f f6      (!u						
0020       aa       96       06       c8       06       fb       00       ae       dc       8b       fd       b8       33       7       91       59	0000	08 00 27 a7 7e 0e 08 00 27 66 25 2b 0	8 00 45 00'.~ 'f?	έ+Ε.		
0030 bd 63 62 0c 39 22 fd bf b9 a7 b6 b8 c1 53 4a 52	0010	00 c2 04 28 00 00 80 11 21 75 0a 00 0	0 02 5f f6( !u.			
0040       ab a4 d4 52 fe e6 90 5e 6e 41 62 9b 26 85 01 b8      ^h nAb.&         0050       45 7c f0 81 e1 22 3f bb ea 09 d8 b6 8b db 7e bc       E ?         0060       e6 51 fd 92 da 67 ca 20 cf 3a 10 e5 10 b1 0f cf			3 37 91 59	7.Y		
0050 45 7c f0 81 e1 22 3f bb ea 09 d8 b6 8b db 7e bc E "?~. 0060 e6 51 fd 92 da 67 ca 20 cf 3a 10 e5 10 b1 0f cf?. 0070 2b c4 f9 17 8f 9d 3f d1 a9 49 1a 3c 73 46 07 10 +?. I.<5F 0080 e8 1d ea 0d 20 30 16 fe df e9 6f e1 26 75 1d 55 0o.&u.U			1 53 4a 52 .cb.9"			
0060 e6 51 fd 92 da 67 ca 20 cf 3a 10 e5 10 b1 0f cf .0g 0070 2b c4 f9 17 8f 9d 3f d1 a9 49 1a 3c 73 46 07 10 +?I. <sf 0080 e8 1d ea 0d 20 30 16 fe df e9 6f e1 26 75 1d 55 0o.&amp;u.U</sf 						
0070 2b c4 f9 17 8f 9d 3f d1 a9 49 1a 3c 73 46 07 10 +?I. <sf 0080 e8 1d ea 0d 20 30 16 fe df e9 6f e1 26 75 1d 55 0o.&amp;u.U</sf 						
	0070	2b c4 f9 17 8f 9d 3f d1 a9 49 1a 3c 7	3 46 07 10 +?I.	<sf< td=""><td></td><td></td></sf<>		
M File: "C:\analyses\uploads\net_14091916430 Packets: 1074 · Displayed: 1074 (100.0%) · Load time: 0:00.0 Profile: Default	0080	e8 1d ea 0d 20 30 16 fe df e9 6f e1 2	6 75 1d 55 0	.&u.U		
	) 🕅	File: "C:\analyses\uploads\net_14091916430 Packets: 10	074 · Displayed: 1074 (100.0%) · Load t	ime: 0:00.0 P	Profile: Default	

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.



November 2014

		Display filter: nor	ne					
otocol	% Packets	Packets	% Bytes	Bytes	Mbit/s E	nd Packets E	ind Bytes E	nd Mbit
Frame	100.00 %	1074	100.00 %	139985	0.003	0	0	0.0
Ethernet	100.00 %	1074	100.00 %	139985	0.003	0	0	0.0
Internet Protocol Version 4	100.00 %	1074	100.00 %	139985	0.003	0	0	0.0
🖃 User Datagram Protocol	19.93 %	214	16.15 %	22602	0.001	0	0	0.0
Domain Name Service	18.06 %	194	13.27 %	18572	0.000	194	18572	0.0
Data	1.86 %	20	2.88 %	4030	0.000	20	4030	0.0
Transmission Control Protocol	78.21 %	840	80.58 %	112793	0.003	656	39352	0.0
Hypertext Transfer Protocol	17.13 %	184	52.4 <mark>5</mark> %	73441	0.002	92	30937	0.0
Line-based text data	8.57 %	92	30.36 %	42504	0.001	92	42504	0.0
Internet Control Message Protocol	1.86 %	20	3.28 %	4590	0.000	20	4590	0.0

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.

_		p [Wireshark 1.10.8 (v1.10.8-2-				
<u>F</u> ile	<u>E</u> dit <u>V</u> iew <u>G</u> o	<u>Capture</u> <u>Analyze</u> <u>Statistics</u>	Telephony <u>T</u> ools <u>I</u> ntern	als <u>H</u> elp		
0	0 📕 📕 🖉	🕒 🖹 🗶 🔁 🔍 🔶	🁒 🌳 ዥ 生 🗐		2, 0, 🖭   🕍 🖄 畅 %   💢	
Filter	:		▼ E	pression	Clear Apply Save	
No.	Time	Source	Destination	Protocol	Length Info	
	31 32.284340		93.177.174.224	UDP	191 Source port: street-stream Desi	
	32 32.284403	93.177.174.224	10.0.0.2	ICMP	219 Destination unreachable (Port u	nreachable)
	33 39.252942	10.0.0.2	199.30.90.80	UDP	302 Source port: street-stream Desi	tination port
	34 39.253080	199.30.90.80	10.0.0.2	ICMP	330 Destination unreachable (Port u	hreachable)
	35 48.144616	10.0.0.2	85.108.52.208	UDP	150 Source port: street-stream Des	cination port
	36 48.144723	85.108.52.208	10.0.0.2	ICMP	178 Destination unreachable (Port u	hreachable)
	37 55.894942	10.0.0.2	201.209.207.224	UDP	298 Source port: street-stream Des	cination port
	38 55.895087	201.209.207.224	10.0.0.2	ICMP	326 Destination unreachable (Port u	hreachable)
	39 61.707488	10.0.0.2	83.28.190.7	UDP	234 Source port: street-stream Desi	cination port
	40 61.707674	83.28.190.7	10.0.0.2	ICMP	262 Destination unreachable (Port un 262 Destination unreachable)	hreachable)
	41 68.832499	10.0.0.2	109.193.194.29	UDP	148 Source port: street-stream Desi	cination port
	42 68.832612	109.193.194.29	10.0.0.2	ICMP	176 Destination unreachable (Port u	hreachable)
	43 75.426646	10.0.0.2	46.49.36.20	UDP	256 Source port: street-stream Desi	cination port
	44 75.426777	46.49.36.20	10.0.0.2	ICMP	284 Destination unreachable (Port u	hreachable)
	45 81.316279	10.0.0.2	200.91.49.183	UDP	267 Source port: street-stream Desi	tination port
	46 81 316401	200.91.49.183	10.0.0.2	ICMP	295 Destination unreachable (Port u	nreachable)

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 && licmp



#### November 2014

Time 22.826025 26.257530 32.284340 39.252942	Source 10.0.0.2 10.0.0.2 10.0.0.2	Destination 8.8.8.8 180.247.156.110	Protocol L DNS	Length Info	0				
26.257530 32.284340	10.0.0.2		DNS	77 C+					
32.284340		180.247.156.110			andard	l quer	y 0x74ce Acr	1.microsoft.com	
	10.0.0.2		UDP	206 So	ource p	ort:	street-stream	Destination port:	24609
20 252042		93.177.174.224	UDP	191 So	ource p	ort:	street-stream	Destination port:	xrpc-registry
39.232942	10.0.0.2	199.30.90.80	UDP	302 So	ource p	ort:	street-stream	Destination port:	7761
48.144616	10.0.0.2	85.108.52.208	UDP	150 So	burce p	ort:	street-stream	Destination port:	4627
55.894942	10.0.0.2	201.209.207.224	UDP	298 50	ource p	ort:	street-stream	Destination port:	qmvideo
61.707488	10.0.0.2	83.28.190.7	UDP	234 So	ource p	ort:	street-stream	Destination port:	12498
68.832499	10.0.0.2	109.193.194.29	UDP	148 SO	burce p	ort:	street-stream	Destination port:	7057
75.426646	10.0.0.2	46.49.36.20	UDP	256 SO	ource p	ort:	street-stream	Destination port:	9752
81.316279	10.0.0.2	200.91.49.183	UDP	267 50	ource p	ort:	street-stream	Destination port:	7399
86.377774	10.0.0.2	84.59.131.0	UDP	144 So	burce p	ort:	street-stream	Destination port:	7605
92.159005	10.0.0.2	94.240.216.82	UDP	181 So	ource p	ort:	street-stream	Destination port:	queueadm
98.129072	10.0.0.2	108.74.172.39	UDP	164 So	ource p	ort:	street-stream	Destination port:	qsoft
105.535522	10.0.0.2	107.193.222.108	UDP	162 SO	ource p	ort:	street-stream	Destination port:	starfish
114.191200	10.0.0.2	94.240.224.115	UDP	155 SO	burce p	ort:	street-stream	Destination port:	8696
121.349339	10.0.0.2	107.217.117.139	UDP	246 SO	ource p	ort:	street-stream	Destination port:	8593
127.551482	10.0.0.2	76.226.114.217	UDP	119 So	ource p	ort:	street-stream	Destination port:	snaresecure
134.176071	10.0.0.2	201.209.58.176	UDP	162 SO	ource p	ort:	street-stream	Destination port:	14191
142.082792	10.0.0.2	95.246.170.150	UDP	208 50	ource p	ort:	street-stream	Destination port:	funk-license
150.253787	10.0.0.2	123.238.67.140	UDP	269 50	ource p	ort:	street-stream	Destination port:	4636
157.348625	10.0.0.2	5.20.67.209	UDP						
	10.0.0.2	8.8.8.8	DNS						
	48.144616 55.894942 61.707488 68.832499 75.426646 81.316279 86.377774 92.159005 98.129072 105.535522 114.191200 121.349339 127.551482 134.176071 142.082792 150.253787 157.348625 164.431849	$\begin{array}{rrrr} 48.144616 & 10.0.0.2 \\ 55.894942 & 10.0.0.2 \\ 61.70748 & 10.0.0.2 \\ 68.832499 & 10.0.0.2 \\ 75.426646 & 10.0.0.2 \\ 81.316279 & 10.0.0.2 \\ 86.37774 & 10.0.0.2 \\ 92.159005 & 10.0.0.2 \\ 98.129072 & 10.0.0.2 \\ 105.535522 & 10.0.0.2 \\ 114.191200 & 10.0.0.2 \\ 114.191200 & 10.0.0.2 \\ 121.349339 & 10.0.0.2 \\ 127.551482 & 10.0.0.2 \\ 134.176071 & 10.0.0.2 \\ 134.176071 & 10.0.0.2 \\ 150.253787 & 10.0.0.2 \\ 150.253787 & 10.0.0.2 \\ 157.348625 & 10.0.2 \\ \end{array}$	$\begin{array}{rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr$	48.144616         10.0.0.2         85.108.52.208         UDP           55.894942         10.0.0.2         201.209.207.224         UDP           61.707488         10.0.0.2         83.28.190.7         UDP           68.82499         10.0.0.2         109.193.194.29         UDP           75.426646         10.0.0.2         46.49.36.20         UDP           83.37774         10.0.0.2         200.91.49.183         UDP           98.13907         10.0.0.2         94.240.216.82         UDP           92.159005         10.0.0.2         107.193.222.108         UDP           105.53552         10.0.0.2         94.240.216.82         UDP           114.191200         10.0.2         94.240.224.115         UDP           124.349339         10.0.0.2         76.226.114.217         UDP           124.349339         10.0.0.2         76.226.114.217         UDP           124.176071         10.0.0.2         201.209.58.176         UDP           142.082792         10.0.0.2         95.246.170.150         UDP           150.253787         10.0.0.2         123.238.67.140         UDP           157.348625         10.0.0.2         8.8.8.8         DNS	48.144616         10.0.0.2         85.108.52.208         UDP         150 56           55.894942         10.0.0.2         201.209.207.224         UDP         298 56           61.707488         10.0.0.2         83.28.190.7         UDP         234 56           68.832499         10.0.0.2         109.193.194.29         UDP         236 56           75.426646         10.0.0.2         46.49.36.20         UDP         256 56           81.316279         10.0.0.2         200.91.49.183         UDP         267 56           86.37774         10.0.0.2         94.240.216.82         UDP         181 56           92.159005         10.0.0.2         107.193.222.108         UDP         164 56           105.535522         10.0.0.2         94.240.224.115         UDP         164 56           105.535522         10.0.0.2         107.193.222.108         UDP         164 56           125.349339         10.0.0.2         76.226.114.217         UDP         155 56           124.191200         10.0.0.2         76.226.114.217         UDP         145 56           124.349339         10.0.0.2         201.209.58.176         UDP         268 56           124.16671         10.0.0.2         201.209.58.176         UDP	48.144616       10.0.0.2       85.108.52.208       UDP       150 source p         55.894942       10.0.0.2       201.209.207.224       UDP       298 source p         61.707488       10.0.0.2       83.28.190.7       UDP       234 source p         68.82499       10.0.0.2       109.193.194.29       UDP       148 source p         75.42664       10.0.0.2       46.49.36.20       UDP       256 source p         81.316279       10.0.0.2       200.91.49.183       UDP       267 source p         86.377774       10.0.0.2       94.240.216.82       UDP       181 source p         92.159005       10.0.0.2       107.193.222.108       UDP       162 source p         105.53552       10.0.0.2       107.217.117.139       UDP       165 source p         114.191200       10.0.0.2       94.240.224.115       UDP       155 source p         121.349339       10.0.0.2       76.226.114.217       UDP       145 source p         124.176071       10.0.0.2       201.209.58.176       UDP       162 source p         134.176071       10.0.0.2       201.209.58.176       UDP       208 source p         142.082792       10.0.0.2       123.238.67.140       UDP       208 source p <tr< td=""><td>48.144616       10.0.0.2       85.108.52.208       UDP       150 Source port:         55.894942       10.0.0.2       201.209.207.224       UDP       298 Source port:         61.707488       10.0.0.2       83.28.190.7       UDP       234 Source port:         68.832499       10.0.0.2       109.193.194.29       UDP       148 Source port:         81.316279       10.0.0.2       46.49.36.20       UDP       256 Source port:         86.37774       10.0.0.2       84.59.131.0       UDP       148 Source port:         86.37774       10.0.0.2       94.240.216.82       UDP       164 Source port:         98.129072       10.0.0.2       107.193.222.108       UDP       164 Source port:         105.535521       10.0.0.2       94.240.224.115       UDP       165 Source port:         114.191200       10.0.0.2       94.240.224.115       UDP       155 Source port:         124.349339       10.0.0.2       76.226.114.217       UDP       195 Source port:         124.176071       10.0.0.2       201.209.88.176       UDP       162 Source port:         134.176071       10.0.0.2       201.209.88.176       UDP       208 Source port:         142.082792       10.0.0.2       123.238.67.140       UDP&lt;</td><td>48.144616       10.0.0.2       85.108.52.208       UDP       150 Source port: street-stream         55.894942       10.0.0.2       201.209.207.224       UDP       298 Source port: street-stream         61.707488       10.0.0.2       83.28.190.7       UDP       234 Source port: street-stream         68.82499       10.0.0.2       109.193.194.29       UDP       148 Source port: street-stream         81.316279       10.0.0.2       20.91.49.183       UDP       267 Source port: street-stream         86.37774       10.0.0.2       94.240.216.82       UDP       181 Source port: street-stream         86.37774       10.0.0.2       108.74.172.39       UDP       164 Source port: street-stream         98.129072       10.0.0.2       107.193.222.108       UDP       165 Source port: street-stream         114.191200       10.0.0.2       94.240.224.115       UDP       155 Source port: street-stream         121.349339       10.0.0.2       107.217.117.139       UDP       246 Source port: street-stream         124.76071       10.0.0.2       76.226.114.217       UDP       119 Source port: street-stream         134.176071       10.0.0.2       95.246.170.150       UDP       208 Source port: street-stream         142.082792       10.0.0.2       123.238.67.140</td><td>48.144616       10.0.0.2       85.108.52.208       UDP       150 source port: street-stream       Destination port:         55.894942       10.0.0.2       201.209.207.224       UDP       298 Source port: street-stream       Destination port:         61.70748       10.0.0.2       83.28.190.7       UDP       234 Source port: street-stream       Destination port:         68.832499       10.0.0.2       83.28.190.7       UDP       248 Source port: street-stream       Destination port:         68.832499       10.0.0.2       109.193.194.29       UDP       148 Source port: street-stream       Destination port:         75.42664       10.0.0.2       260.91.49.183       UDP       266 Source port: street-stream       Destination port:         81.316279       10.0.0.2       94.240.216.82       UDP       144 Source port: street-stream       Destination port:         92.159005       10.0.0.2       108.74.172.39       UDP       164 Source port: street-stream       Destination port:         114.191200       10.0.0.2       94.240.216.82       UDP       155 Source port: street-stream       Destination port:         127.551482       10.0.0.2       70.717.132       UDP       162 Source port: street-stream       Destination port:         127.551482       10.0.0.2       107.217.117.139</td></tr<>	48.144616       10.0.0.2       85.108.52.208       UDP       150 Source port:         55.894942       10.0.0.2       201.209.207.224       UDP       298 Source port:         61.707488       10.0.0.2       83.28.190.7       UDP       234 Source port:         68.832499       10.0.0.2       109.193.194.29       UDP       148 Source port:         81.316279       10.0.0.2       46.49.36.20       UDP       256 Source port:         86.37774       10.0.0.2       84.59.131.0       UDP       148 Source port:         86.37774       10.0.0.2       94.240.216.82       UDP       164 Source port:         98.129072       10.0.0.2       107.193.222.108       UDP       164 Source port:         105.535521       10.0.0.2       94.240.224.115       UDP       165 Source port:         114.191200       10.0.0.2       94.240.224.115       UDP       155 Source port:         124.349339       10.0.0.2       76.226.114.217       UDP       195 Source port:         124.176071       10.0.0.2       201.209.88.176       UDP       162 Source port:         134.176071       10.0.0.2       201.209.88.176       UDP       208 Source port:         142.082792       10.0.0.2       123.238.67.140       UDP<	48.144616       10.0.0.2       85.108.52.208       UDP       150 Source port: street-stream         55.894942       10.0.0.2       201.209.207.224       UDP       298 Source port: street-stream         61.707488       10.0.0.2       83.28.190.7       UDP       234 Source port: street-stream         68.82499       10.0.0.2       109.193.194.29       UDP       148 Source port: street-stream         81.316279       10.0.0.2       20.91.49.183       UDP       267 Source port: street-stream         86.37774       10.0.0.2       94.240.216.82       UDP       181 Source port: street-stream         86.37774       10.0.0.2       108.74.172.39       UDP       164 Source port: street-stream         98.129072       10.0.0.2       107.193.222.108       UDP       165 Source port: street-stream         114.191200       10.0.0.2       94.240.224.115       UDP       155 Source port: street-stream         121.349339       10.0.0.2       107.217.117.139       UDP       246 Source port: street-stream         124.76071       10.0.0.2       76.226.114.217       UDP       119 Source port: street-stream         134.176071       10.0.0.2       95.246.170.150       UDP       208 Source port: street-stream         142.082792       10.0.0.2       123.238.67.140	48.144616       10.0.0.2       85.108.52.208       UDP       150 source port: street-stream       Destination port:         55.894942       10.0.0.2       201.209.207.224       UDP       298 Source port: street-stream       Destination port:         61.70748       10.0.0.2       83.28.190.7       UDP       234 Source port: street-stream       Destination port:         68.832499       10.0.0.2       83.28.190.7       UDP       248 Source port: street-stream       Destination port:         68.832499       10.0.0.2       109.193.194.29       UDP       148 Source port: street-stream       Destination port:         75.42664       10.0.0.2       260.91.49.183       UDP       266 Source port: street-stream       Destination port:         81.316279       10.0.0.2       94.240.216.82       UDP       144 Source port: street-stream       Destination port:         92.159005       10.0.0.2       108.74.172.39       UDP       164 Source port: street-stream       Destination port:         114.191200       10.0.0.2       94.240.216.82       UDP       155 Source port: street-stream       Destination port:         127.551482       10.0.0.2       70.717.132       UDP       162 Source port: street-stream       Destination port:         127.551482       10.0.0.2       107.217.117.139

Figure 87. Wireshark after applying view filter.

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

H 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, <mark>Src Port: street-stream (1736)</mark>, 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: strpe-registry (3651)

Bata (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	e 08 00	27 66 2	25 2b 0	8 00 4	45 00	'.~ 'f%+E.
0010	00 c0 04	17 00 00	0 80 11	da ae (	0a 00 0	0 02 H	b4 f7	
0020	9c 6e 06	c8 60 21	L 00 ac	34 a2 🚦	aa cc 9	f bf :	39 b3	.n`! 4. <mark>9.</mark>
0030	24 42 a2	6f 4d 22	2 3a 82		34 3f 9			\$B.OM":. h-4?.2X>
0040	11 b9 a7	64 60 9	e d1 ea	6b 20 7	75 56 b	6 e0 🛛	82 54	d` k uvт
0050	9d 4a 40	c8 da a	c b8 fd	ca 6d a	ab cb f	a 5e 2	2f ce	.]@m^/.
0060	82 1b 95	a8 42 7	3 a7 30	a7 e4 2	29 c7 7	0 5b 4	4a 97	Bs.0).p[J.
0070	01 22 09	83 ef at	F 86 95	c1 66 2	2b 11 4	1 52 8	80 f9	."
0080	55 66 03	e3 c7 18	8 ae 46	73 61 2	22 44 1	4 84 2	2d 6a	UfF sa"Dj
0090	39 07 a5	ca 71 11	F 5d 80	b7 ed 6	63 65 5	5 3a 1	76 01	9q.]ceU:v.
00a0	88 fe 51	85 37 98	a 41 62	cb c7 d	de 3b 4	f f0 (	OF 07	Q.7.Ab;O
00b0	91 13 50	cf 57 58	8 94 6f	4b f2 d	d5 29 e	42b (	62 73	Р.WX.о К).+bs
00c0	46 6f dc	13 d7 9	d 3a a9	46 61 k	b0 82 d	7 36		Fo: Fa6

Figure 90. Payload of datagram addressed to 180.247.156.110.



Artifact analysis training material

November 2014

0000 0010 0020	00 b1	04 18	00 00	80 11	1f 91	0a 00	00 0	0 45 00 2 5d b1 7 e7 42	'.~ 'f%+E. ]. CI^7.B
0030								5 10 38	{+!e.8
0040	b7 63 🗸	46 53	b9 db	57 77	a7 f8	66 00	f5 a	8 28 e8	.cFsWwf(.
0050	75 eb	cb 6a	1c b5	a8 77	39 4d	04 80	c1 e	8 98 a8	ujw 9M
0060	61 8f	2d ce	66 7d	47 f7	cd 41	67 09	) b4 5 <sup>.</sup>	F 94 47	af}GAgG
0070	6b 5c	33 60	c2 c9	13 d8	d6 b4	30 97	50 d	2 8c 53	k∖3`Õ.PS
0080	20 2d	b3 Of	6e 5b	00 a2	6e ad	62 3f	794	b 43 a4	n[ n.b?уКС.
0090	85 48	92 8c	c1 a1	15 1f	b6 ac	a5 fo	70 a	6 45 a9	.Hp.E.
00a0	35 a5	5b 4d	25 4b	83 29	15 71	4a 60	4e c	4 6c 06	5.[M%K.) .qj`N.l.
00b0	49 45	d3 a9	fa b8	12 3d	a4 32	61 2a	bf O	8 7 b	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*.

Endpoints: net_14	4091916430	)1.pcap						- C X
Ethernet: 3 Fibre	Channel I	FDDI IPv4	:23 IPv6 IPX	JXTA NCP	RSVP SCT	P TCP To	ken Ring UDP: 12	USB WLAN
IPv4 Endpoints - Filter: ip.src == 10.0.0.2 && udp && !icmp								
Address   Packets  Address  Ad								
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	-	-
Name resolutio	Name resolution							
<u>H</u> elp <u>C</u> opy Map <u>C</u> lose								

#### 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.

	_			01.pca													))]	×
En	e <u>c</u> u		-				_		_							_		
0	۲			Ø			X			୍ଦ୍	4	\$		T	₹		] 🖪   🗨 Q, 🔍 📅   🎬 🗹 🅵 %   🧱	
Filt	ter:															•	Expression Clear Apply Save	
No.		Tim	ne			Sou	irce			Dest	inati	on	Pr	otoco	Ler	ngth	Info	
	7	7 16	4.7	8494	8	10	.0.	0.2		10.	0.0	).1		CP		60	49232 > http [FIN, ACK] Seg=272 Ack=409 Win=65292 Len=0	
				8520			.0.			10.				CP		54	http > 49232 [FIN, ACK] Seg=409 Ack=273 Win=30336 Len=0	
	7	9 16	4.7	85594	1	10	.0.	0.2		10.	0.0	).1	Т	СР		60	49232 > http [ACK] Seg=273 Ack=410 Win=65292 Len=0	
	8	0 16	6.2	8780	7	10	.0.	0.2		8.8	8.8.	8	D	NS		91	Standard guery 0x8b0e A zhdxoirxzpvwbyivdaejbwcycg.info	
	8	1 16	6.3	1482	5	8.	8.8	. 8		10.	0.0	).2	D	NS	1	L07	Standard query response 0x8b0e A 10.0.0.1	
	8	2 16	6.3	1615	7	10	.0.	0.2		10.	0.0	).1	T	СР		66	49233 > http [SYN] Seq=0 win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1	
	8	3 16	6.3	1621	9	10	.0.	0.1		10.	0.0	).2	Т	СР		66	http > 49233 [SYN, ACK] Seq=0 Ack=1 win=29200 Len=0 MSS=1460 SACK_PE	F
	8	4 16	6.3	1692	)	10	.0.	0.2		10.	0.0	).1	Т	СР		60	49233 > http [ACK] Seq=1 Ack=1 Win=65700 Len=0	
	8	5 16	6.3	1695	7	10	.0.	0.2		10.	0.0	).1	H	ттр		342	GET / HTTP/1.1	
	8	6 16	6.3	1697	3	10	.0.	0.1		10.	0.0	).2	T	СР		54	http > 49233 [ACK] Seq=1 Ack=289 Win=30336 Len=0	
	8	7 16	6.5	0891	8	10	.0.	0.1				).2	H	ттр	4	162	HTTP/1.1 200 OK (text/html)	
	8	8 16	6.5	0986	5	10	.0.	0.2		10.	0.0	).1	T	CP		60	49233 > http [FIN, ACK] Seq=289 Ack=409 win=65292 Len=0	
	8	9 16	6.5	1017	8	10	.0.	0.1		10.	0.0	).2	T	CP		54	http > 49233 [FIN, ACK] Seq=409 Ack=290 win=30336 Len=0	
	9	0 16	6.5	1068	L	10	.0.	0.2		10.			T	СР		60	49233 > http [ACK] Seq=290 Ack=410 Win=65292 Len=0	
	9	1 16	8.0	0638	2		.0.			8.8			D	NS		91	Standard query 0x7ad0 A fqduprdabairzxamhzxydozuwbq.com	
	9	2 16	8.0	4698	2		8.8					).2	D	NS	1	L07	Standard query response 0x7ad0 A 10.0.0.1	
				4846			.0.					).1		CP		66		
	-			4853	-	_	.0.					).2	T	CP		66	http > 49234 [SYN, ACK] Seq=0 Ack=1 win=29200 Len=0 MSS=1460 SACK_PE	F
	9	5 16	8.0	4887	3		.0.					).1	T	СР		60	49234 > http [ACK] Seq=1 Ack=1 Win=65700 Len=0	
				4891:			.0.					).1	H	ттр		342	GET / HTTP/1.1	
				4892	-		.0.					).2		CP		54	http > 49234 [ACK] Seq=1 Ack=289 Win=30336 Len=0	
	_			0715			.0.					).2		ТТР			HTTP/1.1 200 ОК (text/html)	
	9	9 16	8.3	0840	9	10	.0.	0.2		10.	0.0	0.1	T	CP		60	49234 > http [FIN, ACK] Seq=289 Ack=409 win=65292 Len=0	
< [					-		-				-		111				• • • • • • • • • • • • • • • • • • • •	
-	Enan	0 6	• 6	5 byd	.05	on	win	o (1	28	hi	te)	6	5 b	tos	car	tur	ed (528 bits)	
																	e), Dst: CadmusCo_66:25:2b (08:00:27:66:25:2b)	4
																	. Dst: 10.0.0.2 (10.0.0.2)	,
				27 66										3 00			'f%+'.~E.	
				00 50													P.NS=IZ	
				L4 29	00	00	02	04	05	5 b4	1 0	1 01	. 04	4 02	01	03	r)	
00	40	03 (	07															
	W 🖬	ile: "(	∿\an	alvses)	unlo	/she	net 1	4091	164	30	P:	acket	s: 10	74 - Di	splav	ed: 10	074 (100.0%) · Load time: 0:00.046 Profile: Default	

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

Filter:	p.src==10.0.0.2 &&	dns			•	Expression	Clear	Apply	Save	
o	Time	Source	Destination	Protocol	Length	Info				a mpag
	8 22.826025	10.0.0.2	8.8.8.8	DNS	77	Standard				A crl.microsoft.com
	9 164.431849	10.0.0.2	8.8.8.8	DNS	74	Standard				A www.google.com
8	0 166.287807	10.0.0.2	8.8.8.8	DNS	91	Standard				A zhdxoirxzpvwbyivdaejbwcycq.info
9	1 168.006382	10.0.0.2	8.8.8.8	DNS	91	Standard	query	, y 0x7a	ad0	A fqduprdabairzxamhzxydozuwbq.com
10	2 169.804733	10.0.0.2	8.8.8.8	DNS	85	Standard	query	y 0x23	87d	A qgdcalzlldewcustcdddso.ru
11	3 171.647132	10.0.0.2	8.8.8.8	DNS	92	Standard	query	y Ox5f	16	A dqrktivypwohvgucpwcskizyrcnf.biz
124	4 173.379533	10.0.0.2	8.8.8.8	DNS	90	Standard	query	y 0x54	d4	A nryttbqaegavgjnxrpzhizdgi.info
13	5 175.210218	10.0.0.2	8.8.8.8	DNS	92	Standard	query	y 0x55	ib4	A tgqkdmqgjznfqpvkvytayzxdmpeq.org
14	6 176.943275	10.0.0.2	8.8.8.8	DNS	92	Standard	query	y 0x59	)7a	A pnaqaed1kzaurgtwgmnfugxwufiz.net
	7 178.772607	10.0.0.2	8.8.8.8	DNS	88	Standard				A wgxoprswvsjvojvsbaljblxs.com
	8 180.554605	10.0.0.2	8.8.8.8	DNS	90	Standard				A qolmvhqjzkvfiljsctohykfonba.ru
	9 182.364737	10.0.0.2	8.8.8.8	DNS	93	Standard				A ofobeuylmnwctgtggvcrcamfehovu.com
	0 184.193578	10.0.0.2	8.8.8.8	DNS	91	Standard				A bapzibrkqouoxwygqutwdytvsrs.net
	1 186.035801	10.0.0.2	8.8.8.8	DNS	92	Standard				A kvgtkljkvauempfuxgnfemothemv.org
	2 187.896583	10.0.0.2	8.8.8.8	DNS	86	Standard	query	y Oxe2	2d8	A rdlvpfygyzypytyfmscmv.info
	3 189.725980	10.0.0.2	8.8.8.8	DNS	90	Standard				A cegqwsmnuhllrxmrhqxypozytp.biz
	4 191.598553	10.0.0.2	8.8.8.8	DNS	88	Standard				A dmvpeqyhupjlzwkdifulznfqc.ru
	5 193.365664	10.0.0.2	8.8.8.8	DNS	91	Standard				A dmvcthivmreydlzgaamgyhekbro.com
	6 195.208872	10.0.0.2	8.8.8.8	DNS	92	Standard				A gmovwkdqozdijijrxzxkftsfejv.info
26	7 197.038931	10.0.0.2	8.8.8.8	DNS	86	Standard	query	y 0x54	94	A vkbimpbeiussrirmjhwsca.org

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

Filte	r: 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 HTT	D traffic )	Nirocha	rk filtor

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.

Collow TCP Stream	- • •
Stream Content	
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; S CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; . Host: zhdxoirxzpwbgivdaejbwcycq.info Connection: Close	
HTTP/1.1 200 OK Content-Type: text/html Connection: Close Date: Fri, 19 Sep 2014 14:45:55 GMT Server: INetSim HTTP Server Content-Length: 258	
<pre><html>     <html>         <htmll <html<="" <htmll="" td=""><td>fake mode.<!--</td--></td></htmll></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></html></pre>	fake mode. </td
<pre>p&gt; cp align="center"&gt;This file is an HTML document.   </pre>	
Entire conversation (696 bytes)	•
Eind Save As Print O ASCII O EBCDIC O Hex Dump O C Arrays	Raw
Help         Filter Out This Stream	<u>C</u> lose

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



 	1
\$ cat domains.list   cut -d '.' -f 2   sort   uniq -c   sort -n	l
20 net	ļ
20 org	ļ
21 info	
25 biz	
25 ru	
41 com	ļ

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 D	GA domain name length distribution
	omains.list   cut -d '.' -f 1   awk '{ print length }'   sort iq -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?

November 2014

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 I6XIE6749M.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/ I6XIE6749M.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.



Artifact analysis training material



November 2014

>> 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=0000000000000000000000000831c7cf&a=2
← 200 text/html 258B 140.45kB/s
GET http://afferdls.cn/stat2.php?w=30000&i=0000000000000000000000000000000
← 200 text/html 258B 315.26kB/s
GET http://afferdls.cn/stat2.php?w=30000&i=00000000000000000000000009831c7cf&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=0000000000000000000000000000000
← 200 text/html 258B 444.7kB/s
GET http://afferdls.cn/stat2.php?w=30000&i=0000000000000000000000000000000
← 200 text/html 258B 322.37kB/s
GET http://afferdls.cn/stat2.php?w=30000&i=0000000000000000000000000000000
← 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: Accept: Accept-Language: User-Agent: Host:	Keep-Alive */*	
No content		
[1/20] [showhos	st] ?: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:

- <u>http://api.hostip.info/country.php</u>
- <a href="http://promos.fling.com/geo/txt/city.php">http://promos.fling.com/geo/txt/city.php</a>





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'.



Request	Response
	afferdls.cn Opera/6 (Windows NT 6.1; ; LangID=409; x86) 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:

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

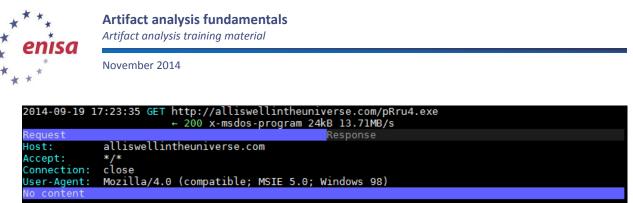


Figure 105. Headers structure of requests from the second group.

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).

INe	etSin	n 💌
	INe	tSim 💽
Т		INetSim 💽
	Т	This is the INetSim default binary
		ОК
		•••

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 17:23:3	4 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.QMifxF.h7ZC2B*.)A/B*
YhXSM@H.@A	m.Nl.{z.,.{%e0HrPG0>#,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'.



November 2014

Request													F	Resp	ons	se			
ost:				favo	ori	tepa	artr	ner.	com	1									
ontent-Len	ath	n:		175															
onnection:			(	clos	se														
ontent-Typ	e:			app	lica	atio	on/o	octe	et-s	tre	eam								
ontent-Enc		ina		bina			,												
ser-Agent:						4/4	Θ (	Com	nat	ibl	ρ:	MS	IF .	5.0	W	indo	ows 98)		
ex						., .			np or c		,						5	[	m:He:
000000000	43	52	59	50	54	45	44	30	94	8e	Θ1	19	a5	3f	45	d2	CRYPTED0?E.		
000000010																	.+oX.QM		
000000020																			
000000030																	ZC2B*.)		
0000000040																	A/B*		
000000040																	\.0.R		
00000000000																	.YhXSM@H.@A.		
0000000070																	m.N1.{z.,.{		
0000000080																	%e0HrPG0.		
000000000000000000000000000000000000000																			
00000000000000000000000000000000000000																a/			
000000000000000000000000000000000000000	эı	12	ZC	89	/C	au	93	17	99	ea	<u>a</u> 9	90	ze	84	1/		,.		
			. 1															1	
13/20] [s	nov	vhos	στj															?:help q	:bac

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:

Analyse the pcap file obtained in the same analysis of *I6XIE6749M.exe* sample. Is there any 1. 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	
	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. Susp	icious Ul	DP 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	тср	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	тср	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	тср	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	тср	54 mit-dov > 49231 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
•					
🕀 Ena	me 16: 66 b	vtes on wir	e (528 bits	). 66 bvt	tes captured (528 bits)

7e:0e (08:00:27:a7:7e:0e)

 ⊞ 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: ⊞ 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:

Filter	: ip.src == 10.0.0.2	&& http			Expression Clear Apply Save
No.	Time	Source	Destination	Protocol	ol Length Info
	6 0.036822	10.0.0.2	10.0.0.1	HTTP	238 GET /country.php HTTP/1.1
	25 1.450394	10.0.0.2	10.0.0.1	HTTP	131 GET /geo/txt/city.php HTTP/1.0
	39 1.789906	10.0.0.2	178.32.190.142	HTTP	223 GET /stat2.php?w=30000&i=000000000000000000000000831c7cf&a=2 HTTP/1.1
	41 1.789965	10.0.0.2	178.32.190.142	HTTP	224 GET /stat2.php?w=30000&i=000000000000000000000000831c7cf&a=99 HTTP/1.1
	47 1.803127	10.0.0.2	178.32.190.142	HTTP	225 GET /stat2.php?w=30000&i=0000000000000000000000009831c7cf&a=100 HTTP/1.1
	70 2.415145	10.0.0.2	10.0.0.1	HTTP	101 GET /rtce007.exe HTTP/1.0
	90 2.780514	10.0.0.2	10.0.0.1	HTTP	101 GET /jucheck.exe HTTP/1.0
1	105 15.006730	10.0.0.2	178.32.190.142	HTTP	224 GET /stat2.php?w=30000&i=000000000000000000000000831c7cf&a=50 HTTP/1.1
	120 16.297630	10.0.0.2	178.32.190.142	HTTP	224 GET /stat2.php?w=30000&i=000000000000000000000000831c7cf&a=22 HTTP/1.1
	122 16.297692	10.0.0.2	178.32.190.142	HTTP	224 GET /stat2.php?w=30000&i=000000000000000000000000831c7cf&a=19 HTTP/1.1
				-	

Figure 111. Connections to external IP address.

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

Starting mitmproxy withouthost flag.	
\$ mitmproxy -n -r mitm.dump	

GET	http://10.0.0.1/geo/txt/city.php
	← 200 text/html 258B 88.4kB/s
GET	http://178.32.190.142/stat2.php?w=30000&i=0000000000000000000000009831c7cf&a=2
	← 200 text/html 258B 140.45kB/s
GET	http://178.32.190.142/stat2.php?w=30000&i=0000000000000000000000009831c7cf&a=100
	← 200 text/html 258B 315.26kB/s
GET	http://178.32.190.142/stat2.php?w=30000&i=0000000000000000000000000000000
	← 200 text/html 258B 128.45kB/s
GET	http://10.0.0.1/rtce007.exe
	← 200 x-msdos-program 24kB 21.21MB/s

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
l   invoice.exe	application/x-dosexec	a4f80b699b52c39da617a6614bb74d9f	i i
<pre>enisa viper &gt; open [*] Session opened l2a8cf772f85eeeac20 enisa viper invoice</pre>	on /opt/viper/projects/0 35eb6c3187650eb9cd7833c79	enisa/binaries/d/9/9/d/d99dfcdd814e 9	ef 39468f 69

Figure 113. Finding invoice.exe sample in Viper

Then send sample to Cuckoo using the Viper cuckoo command.



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.

🕲 Dashboard 🛛 📗	Recent	O Pending	<b>Q</b> Search	🛈 Submit		
cuckoo	T.					
Files URLs	C					
Recent Files					Completed inv	oice.exe analysis
Timestamp			MD5			Status
2014-09-25 11:21:	49		a4f80b69	9b52c39da61	7a6614bb74d9f	reported
2014-09-25 11:17:	23		fa426e8c	d39c44b5002	9f13c0bd645a1	reported
2014-09-25 02:40:	40		a4f80b69	9b52c39da61	7a6614bb74d9f	reported
2014-09-24 16:42:	58		3f0af156	9bee6a34014]	o7aece448193d	reported
2014-09-24 16:40:	39		7d9fcbed	3d6a9124734:	f71ba9ddb2038	reported
2014-09-24 16:38:	19		4166db92	4802cd27246	f5f05fc59ef37	reported

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

To view the analysis report click on md5 sum link.

*** enisa	Artifact analysis funda Artifact analysis training m			
* * *	November 2014			
© Dashboard	ERecent OPending Q Search	<b>⊙</b> Submit		
Quick Overview	Static Analysis Behavioral Analys	sis Network Analysis Dropped Files		
Analysis				
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	
<pre>\$ lab-sendfile /home/enisa/enisa/results/cuckoo1</pre>	

# 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	3
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+hLz7eQrAXMsXU: Olrt0HypbVuhVdvEwQpbVuhVdvEwQpbNAXMsXU: Olrt0HypbVuhVdvEwQpbN+hLz7eQrAXMsXU: Olrt0HypbVuhVdvEwQrAXMsXU: Olrt0H
Yara	None matched
	Download





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

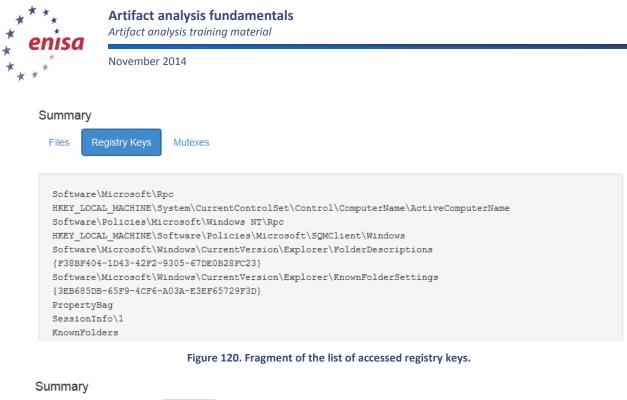
Domain	IP
teredo.ipv6.microsoft.com	
dns.msftncsi.com	
angelescitypattaya.com	
pattayasuay.com	
crl.microsoft.com	
ocsp.msocsp.com	

#### 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\Txxm C:\Users\ENISA\AppData\Roaming\Ixyxm

Figure 119. Fragment of the list of accessed files.



Global\{A760CA8A-5A72-6631-EBE3-86AE005DF631} Local\{36DF8AF0-1A08-F78E-EBE3-86AE005DF631} Global\{3D01DFC6-4F3E-FC50-EBE3-86AE005DF631} Local\{4760F9B4-694C-8631-EBE3-86AE005DF631} Local\{6AFB4B7C-DB84-ABAA-EBE3-86AE005DF631}
Global\{3D01DFC6-4F3E-FC50-EBE3-86AE005DF631} Local\{4760F9B4-694C-8631-EBE3-86AE005DF631}
Local\{4760F9B4-694C-8631-EBE3-86AE005DF631}
Local\{6AFB4B7C-DB84-ABAA-EBE3-86AE005DF631}
Local\{1D79EFC1-7F39-DC28-EBE3-86AE005DF631}
Global\{DA3F107E-8086-1B6E-0ABC-17B7E1026728}
Global\{DA3F107E-8086-1B6E-0EBD-17B7E5036728}
Global\{DA3F107E-8086-1B6E-5EBD-17B7B5036728}
Global\{DA3F107E-8086-1B6E-8EBD-17B765036728}

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.

Quick Overview	Static Analy	sis	Behavioral Analysis	Network Analysis	Dropped Files
Static Analysis	Strings A	ntivir	us		

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<sup>10</sup>. The aim of this structure is to give information about the executable version number, operating system, description, as well as the original file name.

<sup>&</sup>lt;sup>10</sup> 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	0x0000000	0.0
\x87\x17j\x97\xeb\xf1\x0b\x11	0x00036000	0x0002f000	0x0002e800	7.99661984982
.rsrc	0x00065000	0x00001000	0x0000c00	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

NEL32.DLL:			
LoadLibraryA			
GetProcAddress			
VirtualProtect			
VirtualAlloc			
VirtualFree			
ExitProcess			
	RNEL32.DLL: LoadLibraryA GetProcAddress VirtualProtect VirtualAlloc VirtualFree ExitProcess aPI32.dll: CryptSetProvParam	LoadLibraryA GetProcAddress VirtualProtect VirtualAlloc VirtualFree ExitProcess	LoadLibraryA GetProcAddress VirtualProtect VirtualAlloc VirtualFree ExitProcess

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

## 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).



Strings

November 2014

Static Analysis

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.

d99dfcdd814ef3	89468f6912a8cf772f85eeeac285eb	6c3187650eb9cd78	33c79 yhih.exe	taskhost.exe	Dwm.exe
Explorer.EXE	FileZilla Server Interface.exe	Greenshot.exe	cmd.exe		
	yhih.ex	<b>e</b> , PID: <b>3240</b> , Paren	it PID: 2824	Click to fi	Iter calls
	default network filesyste	m registry proce	ess services synch	ronization	

## 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: 0x0000064 DesiredAccess: 15 ObjectAttributes: C: \Sessions \1\BaseNamedObjects	SUCCESS	0x0000000	
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*.



## Processes

registry filesystem process services network synchronization
d99dfcdd814ef39468f6912a8cf772f85eeeac285eb6c3187650eb9cd7833c79 PID: 1540, Parent PID: 3896
unweh.exe PID: 580, Parent PID: 1540
taskhost.exe PID: 1876, Parent PID: 480
Dwm.exe PID: 1740, Parent PID: 796
FileZilla Server Interface.exe PID: 1412, Parent PID: 1764
Explorer.EXE PID: 1764, Parent PID: 1808
Greenshot.exe PID: 852, Parent PID: 1764
cmd.exe PID: 3692, Parent PID: 1540

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.

Quick Overview Static Analy		Analysis I	sis Behavioral Analysis		Network Analysis	Dropped Files	
Download PCAP							
Hosts (2)	Domains (6)	HTTP (8)	ICMP (0)	IRC (0	)		
Hosts							
IP							
8.8.8.8							
239.255.255.2	250						

## 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.



## HTTP Requests

URI	Data			
http://angelescitypattaya.com/mimosa/file.php	<pre>POST /mimosa/file.php HTTP/1.1 Accept: */* User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; W indows NT 6.1; Trident/4.0; SLCC2; .NET CLR 2.0. 50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; M edia Center PC 6.0; .NET4.0C) Host: angelescitypattaya.com Content-Length: 122 Connection: Keep-Alive Cache-Control: no-cache TA\x14\xd9n+\x80\xde\xae@\x8a\x8e&amp;\xe6E)7\xa50g\ x19N\x807\x9f,P\x1f\xc8E\xb4\x93\xe9\xe5\x1dS\x9 f\xe0\xd4\x13\xd1\xca\x9b}\xd2\x85\xf6\xf6[\xa0\ xa3h\xa1\xde\x8e\xd3ui\xf5\xf3\xf6#\xebF\x15\xc2 \xfd\x8c\xc4\x8d\xc4\xf1\x98g\xab\xd7\xdarm-\xe0 \xc7\x80\x8f\x9ft\xd0\xae\x13\xce \xb7\xc6\xc3\x</pre>			
	<pre>ff~\x17BpSi,\x0c\xd4fq\xcc\x13+\xd0\xcf]\xc7\xff \x18\xbem\xc6\x1e\xb6Hm</pre>			

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.



Artifact analysis training material

November 2014

Dropped Files

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: OIJ trgLcG/wpbV9IFztXsWTndvtXSwQ9bn08KbclmZVfn+RNriKeQrAXMsXmH: OIrt0HYpbVuhVdvEwQ9bn+hLz7eQrAXMsXmH: OIrt0HypbVuhVdvEwQpAAMsXmH: OIrt0HypbVAAMsXmH: OIrt0HypbVAAMsXmH: OIrt0H
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

- A Study of the Packer Problem and Its Solutions
   <u>http://www.ecsl.cs.sunysb.edu/tr/TR237.pdf</u> (accessed 15. October 2014)
- 2. Visual Studio 2013 Update 3 <a href="http://www.visualstudio.com/">http://www.visualstudio.com/</a> (accessed 15. October 2014)
- 3. What is a DLL? <u>http://support.microsoft.com/kb/815065</u> (accessed 15. October 2014)
- 4. Base64 http://en.wikipedia.org/wiki/Base64 (accessed 15. October 2014)
- 5. Borland Delphi <u>http://www.on-time.com/rtos-32-docs/rttarget-32/programming-</u> <u>manual/compiling/borland-delphi.htm</u> (accessed 15. October 2014)
- Advanced Techniques Using Process Explorer <u>http://www.microsoft.com/security/sir/strategy/default.aspx#!malwarecleaning\_explorer</u> (accessed 15. October 2014)
- VERSIONINFO resource <u>http://msdn.microsoft.com/en-</u> 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