



SAPIENZA
UNIVERSITÀ DI ROMA

FACULTY OF INFORMATION ENGINEERING, COMPUTER SCIENCE AND STATISTICS

Master Thesis in

ENGINEERING IN COMPUTER SCIENCE

**Reverse Engineering For Malware Analysis:
Dissecting The Novel Banking Trojan ZeusVM**

Candidate

Donato Dell'Atti

Advisor

Prof. Roberto Baldoni

Student ID

1231142

Assistant Advisors

Dott. Leonardo Aniello

Dott. Daniele Ucci

Academic Year 2014/2015

Contents

Abstract.....	vii
1. Introduction.....	1
2. Malware Categories: Purposes and Security Techniques	3
2.1. Malware Categories: Purpose.....	3
2.1.1. Virus	3
2.1.2. Worm	4
2.1.3. Trojan	4
2.1.4. Spyware.....	4
2.1.5. Rootkit.....	4
2.1.6. Botnet	5
2.2. Malware Categories: Security Techniques	5
2.2.1. Encrypted Malware.....	5
2.2.2. Oligomorphic Malware	5
2.2.3. Polymorphic Malware.....	6
2.2.4. Metamorphic Malware	6
2.3. Obfuscation Techniques	6
2.3.1. Dead Code Insertion	6
2.3.2. Register Reassignment.....	7
2.3.3. Subroutine Permutation	7
2.3.4. Instruction Substitution	7
2.3.5. Code Transposition	7
2.3.6. Code Integration	8

3. Reverse Engineering.....9

3.1.	Malware Analysis Techniques.....	10
3.1.1.	Static Analysis	10
3.1.2.	Dynamic Analysis	11
3.2.	Tools for Malware Analysis.....	12
3.2.1.	Hash Algorithm Based Software	12
3.2.2.	Antivirus	13
3.2.3.	Packer Detector	13
3.2.4.	Header and Sections Inspector.....	14
3.2.5.	String Analysis.....	14
3.2.6.	Disassembler.....	15
3.2.7.	Decompiler.....	15
3.2.8.	Debugger.....	15
3.2.9.	Registry Monitor	16
3.2.10.	File System and Process Monitor.....	17
3.2.11.	Network Monitor	17
3.2.12.	Virtual Machine.....	18

4. The Banking Trojan Zeus 19

4.1.	Introduction	19
4.2.	History.....	20
4.3.	Toolkit	23
4.3.1.	Config.txt.....	23
4.3.2.	WebInjects.txt.....	24
4.3.3.	Command & Control Server.....	25
4.3.4.	The Builder	26
4.3.5.	The Executable.....	27

4.4.	How Zeus works.....	27
5.	Reverse Engineering of ZeusVM	29
5.1.	Case Study Environment.....	29
5.1.1.	Creation of the Virtual Machines.....	30
5.1.2.	Installation of the ZeusVM Control Panel.....	30
5.1.3.	Creation of the ZeusVM trojan	31
5.1.4.	Tools Setup.....	32
5.2.	Analysis	32
5.2.1.	Malware testing: Basic Static Analysis.....	33
5.2.2.	Advanced Dynamic Analysis	36
5.2.3.	Static Analysis of the Virtual Machine	36
5.2.4.	Dynamic Analysis	39
5.2.5.	Basic Dynamic Analysis	39
5.2.6.	Dynamic Analysis of Dropped.exe	40
5.2.7.	Dynamic Analysis of RC4 S-Box.....	41
5.2.8.	Static Analysis of RC4 PRNG.....	43
5.2.9.	Remote Debugging of Explorer.exe.....	43
5.2.10.	C&C URL Decryption	45
5.2.11.	DynamicConfig Decryption	46
5.2.12.	Traffic Analysis	47
5.2.13.	Dynamic Analysis of communications	49
5.2.14.	Static Analysis of POST data.....	51
5.2.15.	Multiple Malware Samples Analysis.....	52
5.3.	Summary.....	54
5.3.1.	Missing pieces	56
6.	Conclusions.....	58

6.1. Future Works	58
7. References	60

List of Figures

<i>Figure 1 - Zeus timeline</i>	<i>20</i>
<i>Figure 2 - Toolkit scheme.....</i>	<i>23</i>
<i>Figure 3 - Config.txt.....</i>	<i>24</i>
<i>Figure 4 – Webinject.txt.....</i>	<i>25</i>
<i>Figure 5 – Builder Control Panel</i>	<i>26</i>
<i>Figure 6 – ZeusVM Builder</i>	<i>27</i>
<i>Figure 7 – Environment</i>	<i>30</i>
<i>Figure 8 – ZeusVM decryption overview.....</i>	<i>33</i>
<i>Figure 9 – VirusTotal analysis</i>	<i>33</i>
<i>Figure 10 – PEiD analysis</i>	<i>34</i>
<i>Figure 11 – PEview analysis</i>	<i>35</i>
<i>Figure 12 – BinText analysis.....</i>	<i>35</i>
<i>Figure 13 – Virtual Machine.....</i>	<i>37</i>
<i>Figure 14 – URL Decryption.....</i>	<i>45</i>
<i>Figure 15 – DynamicConfig inside JPG</i>	<i>47</i>
<i>Figure 16 – Communication Bot-C&C</i>	<i>48</i>
<i>Figure 17 – Packet Exchanged Bot-C&C.....</i>	<i>49</i>
<i>Figure 18 – Decrypted POST data</i>	<i>51</i>
<i>Figure 19 – Decrypted StaticConfig.....</i>	<i>52</i>

<i>Figure 20 – VM functions during execution</i>	<i>54</i>
<i>Figure 21 – ZeusVM execution</i>	<i>55</i>
<i>Figure 22 – ZeusVM</i>	<i>56</i>

Abstract

In recent years, Internet Security has acquired a key role in Computer Science due to the huge damages caused by security outbreaks. In particular, during the last decades, there has been a rise of Botnets as a mechanism to steal money and sensitive information.

In this scenario, one of the most important families of Botnet are currently created using the Zeus toolkit, through the diffusion of the Zeus trojan or ZBot that has been firstly discovered in 2007.

In order to fight back these threats, Reverse Engineering has become a standard procedure in Malware Analysis. In this field, Reverse Engineering is applied in order to understand the behaviour of a malware through the reconstruction and analysis of the components of the software source code.

Considering the importance of this topic, this thesis focuses on the ZeusVM v2.0.0.0. trojan, as it is one of the most recent addition to the family of Zeus-based Botnets and a complete version of its Toolkit has been leaked on internet for free in July 2015 allowing everyone to create his or her own botnet.

In this dissertation, aiming to understand ZeusVM trojan behaviour and its security mechanisms against detection and anti-analysis, the process of Reverse Engineering was applied as the source code was not available. This process has been adopted inside the Malware Analysis using the Static and Dynamic techniques, which use both the Reverse Engineering in different ways.

As a result of this analysis, a new technique to decrypt the URL of the Command & Control Server was found. Moreover, the role of specific indexes in the RC4 decryption, and the technique used to encrypt the traffic with the C&C were found.

This allowed the identification and classification depending on their roles of some functions involved with the Virtual Machine during the execution of the malware.

1. Introduction

The name Malware stands for ***Malicious software***, a malicious software is a software that runs in a computer without the knowledge or the agreement of its owner. Different types of malware exist and they can be classified depending on their spreading technique or their purpose, which can range from monetizing the security outbreaks, or gaining valuable information that could be sold, to damaging the hosting machine or overloading the network.

In recent year, considered the yearly increasing spread of new malware and their dangerousness, enhancing the defences against them has become an important need. In this scenario, internet security has acquired a key role in order to protect sensitive information since the most damaging attacks usually involve stealing money.

Nowadays, Botnets are mainly used to carry out these attacks. One of the most important families of Botnet is created using the Zeus toolkit, through the diffusion of the Zeus trojan or ZBot that has been firstly discovered in 2007.

This thesis focuses in particular on the *ZeusVM v2.0.0.0* trojan as it is one of the most recent addition to the family of Zeus based Botnets and a complete version of its Toolkit has been leaked on internet for free in July 2015 allowing everyone to create its own botnet.

As the source code of ZeusVM was not available, in order to understand how it works the process of *Reverse Engineering* was applied.

In Software Engineering, the term *Reverse Engineering* stands for the process of analysing a subject system to create representations of the system at a higher level of abstraction (Chikofsky, et al., 1990).

The aim of this dissertation was to carry out a malware analysis of the malware ZeusVM through a process of *Reverse Engineering* in order to understand its behaviour and its security mechanisms *against detection and Malware Analysis*.

The analysis of the ZeusVM trojan was performed through the use of the *Static* and *Dynamic Analysis*, the main techniques of Malware Analysis. Both these techniques can be further categorized in *Basic* and *Advanced*. The *Basic Analysis* is a superficial analysis which involves the appearance of the malware and its behaviour. The *Advanced Analysis* is the code analysis, where there is a deep inspection of the internals of the malware. The Advanced analysis can be called also *Reverse Code Engineering* and frequently in the Software Engineering sector, it is abbreviated to Reverse Engineering, losing its first initial meaning.

In this dissertation, mostly Reverse Code Engineering was applied to understand the internals of the ZeusVM malware. Virtualization was used to analyse the trojan, a Botnet was deployed to replicate the execution on an infected machine, including the communication with a fictional server acting as the Command & Control. The ZeusVM trojan was analysed starting from its most peculiar characteristic, the Virtual Machine, whose components were found out. Then, a deep analysis of the configuration file of the trojan and its decryption has been carried out as well as the traffic between the infected machine and the server were analysed.

As a result of this analysis, a new technique adopted inside the malware to decrypt the URL of the Command & Control Server used to download the DynamicConfig was found. Moreover, the role of specific indexes in the RC4 decryption, and the use of the VisualEncrypt and RC4 by ZeusVM to encrypt its traffic to the C&C were found. This allowed the identification and classification depending on their roles of some functions involved with the Virtual Machine during the execution of the malware.

Firstly, an initial theoretical study of the malware and of the most common techniques applied in Reverse Engineering and Malware Analysis was carried out as shown in chapter 2, 3 and 4. Secondly, the setup of the environment was created and the analysis of the malware was conducted as explained in chapter 5. Limitations and conclusions are reported in chapter 6.

2. Malware Categories: Purposes and Security Techniques

During the years, Malware have changed their behaviour and purpose since also the writers of the code have changed.

In the early years of internet, there have been many cases where the coders were students who wanted to perform a prank, like the first internet worm, or just gain popularity. Nowadays, the situation has heavily changed and the most powerful malware are written by skilled programmers whose job is to develop them and their purpose ranges from monetizing the security outbreaks gaining valuable information that could be sold, to damaging the hosting machine.

Currently, malware can be categories depending on their purpose and spreading techniques (Damodaran, 2015), such as trojan, worms, spyware, etc. (fig,1), or behaviour in terms of security techniques that they use to avoid detection or to enhance their analysis complexity such as Oligomorphic, Polymorphic malware etc. and related obfuscation techniques (Agarwal, et al., 2013).

In this chapter, the key features of the main purpose-categories and the most common security and related obfuscation techniques are described.

2.1. Malware Categories: Purpose

2.1.1. Virus

Generally, Malware and Virus are considered synonymous but they are not as Virus is a sub-category of Malware. A Virus takes its name from biology, because its behaviour is similar to its biological counterpart. As real viruses, computer viruses need to attach to other programs to live and they self-replicate.

Once infected the machine, the virus replicates itself and infects all the machines connected to the source of the infection. Once infected the system, it is modified and the vital functions to execute programs are destroyed.

Viruses are a primitive form of malware. They were firstly developed in the pre-internet era and the main vehicles of transmission were physical devices such as pen drive, etc... With the advent of internet, they were developed in order to spread through the network. At the same time, new forms of malware such as worms, with more advanced and sophisticated characteristics, were created.

2.1.2. Worm

A Worm is similar to a Virus as it is a self-replicating malware but it differs from a virus, as it does not need to attach to other programs to survive. For this reason, it is defined as a stand-alone program. It mainly propagates through networks.

2.1.3. Trojan

This type of malware takes its name from the wooden horse used to enter Troy during the Troy war. As the Troy horse, a trojan is apparently an innocuous artefact that has access through the front door besides it contains a malicious element hidden inside it. Its definition recalls its characteristics, as its installation requires user's consent. On the contrary, of viruses and worms, it does not self-replicate.

2.1.4. Spyware

This category is relatively recent and refers to that malwares, that as the name itself states, spy the user tracking, monitoring and reporting users' online activities without their consent. These malwares are capable of collecting a wide range of information including cookies, credentials, credit card numbers, etc. They differ depending on how intrusive they are.

2.1.5. Rootkit

Rootkit is not a malicious software but it can be used for malicious activity. Its goal is to hide itself inside the system and provides a privileged access to the system for the

attacker. A rootkit can coexist with other malware and has the role of concealing their malicious activity so that they cannot be detected.

2.1.6. Botnet

A Botnet is a network of infected machine with a particular Bot. A bot is a malicious code that infects a machine connected to internet. This bot allows the owner of the botnet, also known as bot master, to remotely control every machine inside the botnet. The botnet can be used to perform spam activity through emails or through a spyware component that can collect bank credentials or other valuable information.

2.2. Malware Categories: Security Techniques

It is possible to categorize malware in terms of the security techniques that they use to avoid detection or to enhance their analysis complexity. The most common categories are encrypted, oligomorphic, polymorphic and metamorphic malware.

2.2.1. Encrypted Malware

Encryption is a technique that hides the content of a malware from a static analysis, which is an analysis that does not execute the code and does not have the possibility to run the decryption function to decrypt the malware. Once discovered the decryption function, the malware is vulnerable, since it is composed of the encrypted part and of a decryption function, which is always the same. The encryption avoids the detection from a signature based scan. This technique can be combined in multiple levels of different encryption to make the malware more dangerous and less vulnerable.

2.2.2. Oligomorphic Malware

An Oligomorphic malware is essentially a slight modification of an Encrypted malware where the decryption function is not fixed and easily identifiable. For each different sample, a different decryption function is created, this makes the malware always virtually different. In real cases, the combinations of the decryption functions are

limited, so it is possible to take the sign of every different decryption function and identify them through a signature based analysis.

2.2.3. Polymorphic Malware

The Polymorphic malware is an evolution of the Oligomorphic malware. In this case, the code of the malware takes a mutation from its original source and the malware generates a real infinite number of different decryption functions through obfuscation techniques, so each sample is different from the others and needs to be specifically analysed. Besides this, the encrypted malware is always the same as well as the decryption function.

2.2.4. Metamorphic Malware

A Metamorphic Malware is a type of malware which uses the most complex security technique compared to a polymorphic malware. In this case, the malware is rewritten every time but it does not need to use encryption, since all the body of the malware is changed at each rewriting. The functionality of the malware remains the same but through different practices, the outcome is always different. The Malware contains a mutation engine that has the role of creating the new mutated sample.

2.3. Obfuscation Techniques

Code Obfuscation is a legitimate technique used by many software developers to hide the source code of their works or to make it harder to recreate the source code through reverse code engineering. Malware writers adopt the same strategies to hide their malicious software from the researcher.

Those methods are used also in polymorphic and metamorphic malware.

2.3.1. Dead Code Insertion

Dead code insertion is a simple technique that adds some operations that are not needed for the program, like NOP instructions that do not change the behaviour of the program. Antivirus could check for a series of NOP operation and could delete them, so

also other dead code could be inserted to slightly modify the program, like the subsequent increment and decrement of a variable. The dead code could be never executed or even if executed it would have no effects on the functionality of the original form of the malware.

2.3.2. Register Reassignment

Register Reassignment or Renaming is a technique that changes the used register from one generation of the malware to another. The behaviour and the functionality of the program remain untouched. It is to reassign a register that is never used inside the program, otherwise it is a technique more complex to adopt.

2.3.3. Subroutine Permutation

Subroutine Permutation exchanges the order of the malware routine in a random combination. This technique could create a $n!$ possible combinations of a malware with n subroutines. The order of the subroutines could be different for each sample.

2.3.4. Instruction Substitution

Instruction Substitution changes the code of the program with equivalent operations that logically have the same results but that are executed in a different way. This is a complex obfuscation technique since it is needed a dictionary of all the possible substitutions than could occur for each operation to detect it.

2.3.5. Code Transposition

Code transposition is a technique where the order of the instructions is changed from the original source of the malware. Blocks of code that are not dependent, they are rearranged in order to change the resulting code of the malware without changing the behaviour of those blocks of codes. This technique is hard to implement because it is complex to find independent blocks of code. Another possible way to achieve a similar result is to randomly rearrange the instructions and reconstruct the right order inserting conditional and unconditional jumps inside the code.

2.3.6. Code Integration

Code Integration is a sophisticated technique. In this case, the virus inserts its code into another executable program. In order to do it, it firstly disassembles the host program then copies itself inside it and recompiles the program to generate a new executable.

3. Reverse Engineering

Reverse Engineering is the process of extracting information from a software program's binary code by analysing its components and behaviour and without any knowledge about its internals and any information about its creation.

Reverse Engineering is a field that can be applied to every Forward engineering sector, it is comparable to a scientific research with the difference that the analysis is computed on a man-made product and not a natural phenomenon.

In Software Engineering, the term reverse engineering is the process of analysing a subject system to create representations of the system at a higher level of abstraction (Chikofsky, et al., 1990) .

Reverse Engineering is a process opposed to the traditional waterfall model, in which the aim is to produce the source code of the software. It only examines the provided software to gain information without writing the source code even if available.

There are mainly two possible fields in which reverse engineering could be useful (Eilam, 2005).

The first field is in software development where the source code is available but it is poorly documented and there is the needed to do interoperability with this undocumented or proprietary piece of code (Eilam, 2005).

The other main field is related to software security, where the source code is not available and the object of the reverse engineering is to reconstruct the source code of the software analysed (Eilam, 2005).

In Software Security, reverse engineering can be applied also to the research of security flaws inside software, for the analysis of cryptographic algorithms or to break the security layer of software protected by digital rights management.

This field includes Malware Analysis since usually malware developer do not share their source code and as a consequence malware analysts need to understand the behaviour of the malware trying to reconstruct the source code of the desired aspects that they want to analyse.

3.1. Malware Analysis Techniques

Malware Analysis is the study of a malware by dissecting its components to understand its behaviour. There are mainly two possible Malware Analysis techniques, each one has its advantages and disadvantages. These techniques are *Static Analysis* or also called code analysis, and *Dynamic Analysis* or behaviour analysis. Both techniques can also be categorized in *basic* and *advance*, so there are four categories *Basic Static Analysis*, *Advanced Static Analysis*, *Basic Dynamic Analysis* and *Advanced Dynamic Analysis*. (Sikorski, et al., 2012)

3.1.1. Static Analysis

The procedure of analysing code without executing it is called *static analysis*. This is the first analysis that should be taken on an executable to understand its behaviour. *Static Analysis* has the main advantage that does not execute the code of the malware so it is not harmful for the system that runs the analysis (Singhal, et al., 2014). As previously mentioned, *Static Analysis* can be divided in *Basic* and *Advanced*.

Basic Static Analysis

Basic Static Analysis consists of an analysis of the malware without viewing the machine level instruction of the file. There are programs that could be used to gain some information, firstly an antivirus scan could reveals the malicious essence of the file. An hash signature verification could be performed to see if the file is known. The structure of the file could be dissected to see if the program is a portable executable or if it has been packed.

Advanced Static Analysis

Advanced Static Analysis has the role of inspecting the code of the malware with a proper disassembler. An example of disassembler is IDA Pro (Hex-Rays) which stands for Interactive Disassembler Professional and is usually the first choice for malware analysts. It can also be used as a debugger.

Generally, a disassembler is a tool that reconstructs the assembly code of the malware. Through this analysis, it is possible to see all the possible instructions that the malware could execute on the computer. Moreover, it is possible to identify specific function inside the code that has a known implementation or a specific digital signature. For example, it is possible to identify functions that do encryption or perform obfuscation.

The problem with this kind of analysis is that analysing the binary all the data structures and variables are not available, so it is hard to understand the behaviour of the program. (Gandotra, et al., 2014)

This technique has some limitations since malwares have implemented a lot of techniques to hinder this kind of analysis for security researchers, like the obfuscation or the encryption of some parts of the malware, which cannot be read (Gadhiya, et al., 2013).

3.1.2. Dynamic Analysis

Dynamic Analysis is a complementary approach to the Static Analysis, as it is the analysis of a software during its execution in a controlled environment. Since the software could be malicious, and Malware Analysis is about malicious software, the environment where the analysis is taken should be safe. *Dynamic Analysis* is performed under safe environment that cannot infect the machine of the security researcher. This environment is based on Virtual Machine or Sandbox.

The problem with *Dynamic Analysis* is that malware are aware of this technique and have implemented countermeasure, a malware that can identify a hostile ambient could act in a different way from its standard execution.

One of the main problem with *Dynamic Analysis*, it is that it is based on a single execution of a malware, which means that not all the paths of that program can be identified in a single run. Finding all the possible executions of the malware could be time consuming, depending on the nature of the malware.

Basic Dynamic Analysis

Basic Dynamic Analysis is the execution of a malware and the study of its effects on the system. This analysis can be taken with the help of software that monitor the system to see accesses to function calls, the creation of new files, the exchanges of internet packets, the accesses to the register or any possible information that could be gain from the observation of the environment before and after the execution of the malware.

Advanced Dynamic Analysis

Advanced Dynamic Analysis is the execution of the malware with a debugger software such as IDA Pro or OllyDbg (Yuschuk).

A debugger allows the execution of one operation of code at the time, so it possible to do a deep inspection of each function that the malware executes. Through a Dynamic Analysis compare to a Static Analysis, it is possible for the security researcher to find the values assigned to variables during the runtime execution.

The problem with this approach is that many security checks have been implemented inside malware to identify the execution inside a debugger or a virtual machine.

3.2. Tools for Malware Analysis

In this section, an overview of the software that can be used for static and Dynamic Analysis is presented.

3.2.1. Hash Algorithm Based Software

Hashing is a common technique used to identify malware. Malware that does not change their executable maintains a static sign, or fingerprint. This can be checked

with algorithms like Message Digest Algorithm 5 (Rivest, 1992) or Secure Hash Algorithm 1 (Eastlake, et al., 2001) that are the most popular and commonly used algorithm for malware identification.

3.2.2. Antivirus

The main software that can be used to identify a malicious software is an antivirus. Scanning a file with an antivirus could give some initial information about it as the file could be a well-known malware that has some specific characteristics.

For this purpose, there are online services that provide a free scan for uploaded file. Any file can be submitted to a virus company that will test it. The most important and famous online scanner is VirusTotal, but there are also other companies that provide this service like Jotti Malware Scan and many others. Those services provide a report with the result of the analysis on the file.

3.2.3. Packer Detector

The first thing to do with a malware file is to understand if it is an executable. There are a lot of software that can analyse the structure of a portable executable but that software will fail if the software is packed (Sikorski, et al., 2012).

Malware authors use packing technique to hide the content of the executable, packed programs are programs in which the malware has been compressed and it is not possible to analyse the program with a static analysis. A packer always takes in input an executable and outputs an executable that has the same functionality but has been transformed with either encryption or compression techniques. This procedure makes the reverse engineering of that malware more complex. In the past, this technique was used also to reduce the size of the malware, a packer can be applied numerous times to an executable to encrypt it multiple times.

Those kind of malware needs to be unpacked prior to be analysed. There are software such as PEiD (altheid) or Exeinfo PE (A.S.L) that identify if an executable is packed and which is the packer that has been used.

PEiD has been discontinued from its creators but it is still considered the best software in this field (Sikorski, et al., 2012).

In order to avoid the detection from these software, malware authors can implement custom packers. In this case, it could be needed to do a manual unpacking to gain the original form of the malware executable.

3.2.4. Header and Sections Inspector

Almost all the windows executable objects are in the file format of Portable Executable [PE]. PE file has information in its header that could be of great value for malware analysts.

Once unpacked a malware executable, it can be analysed with software that inspect the header of the file and its structures to gain information. One of the tools that can be used is PView (Radburn). Using this software it is possible to identify the structure inside the PE. Usually, four sections are identifiable: *.text*, *.rdata*, *.data*, and *.reloc*. Those information about the sections are written inside the header with also other interesting elements such as import export functions, time of compilation and others.

Usually *.text* section contains the instructions that are executed by the CPU; *.rdata* contains the import/export information, and read-only data used by the program. *.data* contains the program global data, while local data are not stored in this section. *.reloc* section contains information for the relocation of library files.

Actually, the name of the section is relatively important as sometimes it can be obfuscated to make analysis more complex.

There are many software to do the header and section inspection such as PE.Explorer (HeavenTools) and others.

3.2.5. String Analysis

String Analysis is the research of readable text embedded inside a malware. This can help to find some valuable information. Embedded strings could be easily extracted through software like Bintext (McAfee), which has specifically only this role or using

more complex software such as IDA Pro which have this same feature. This kind of analysis is not very useful for a malware, which encrypts its strings.

3.2.6. Disassembler

The Disassembler is one of the most important software in reverse engineering and in particular for the Advanced Static Analysis. The disassembler is a software that takes in input an unreadable executable binary file and generates an output of the same code into, a human readable, assembly language code.

Assembly is machine dependent, since the instruction sets are different from one architecture family to another. The disassembler should be capable of understanding the different types of architecture and adapts to them.

Usually malware are written for Windows x86 architecture, but there also many other architectures x64, ARM, and others that could be a target for malware. The most famous and powerful disassembler is IDA Pro. It usually comes in two versions, and only the most complete and expensive one supports all the architectures.

Another popular free disassembler is OllyDbg but it supports only x86 instructions.

3.2.7. Decompiler

A Decompiler is a software that transforms a series of assembly instructions into a high level of instructions, frequently in a C-like language. It has the role to reconstruct the source code of the analysed application, but actually it does not exist a software which makes a complete reconstruction. The Hex-Rays Decompiler is a plugin for IDA Pro and it is the only one which gives some useful results decompiling portions of code.

3.2.8. Debugger

The Debugger is the most important software for Advanced Dynamic Analysis. It is a computer program that runs another program to study it and detect errors. The debugging phase is a common phase for every software development. The debugger is usually inside the integrated development environment and it is a source level debugger which can take breakpoints directly inside the source code of the software.

In Malware Analysis, the debugger has a slightly different role since the source code is not available, so it is needed an assembly debugger, also called, low level debugger. This kind of debugger works directly on assembly code and allows the researcher to analyse one instruction of the program at a time.

A debugger shows the current state of registers and memory during the execution. Frequently disassembler and debugger are included in the same software, this is the case also of for IDA Pro and OllyDbg.

3.2.9. Registry Monitor

The Windows registry is used to store settings and options of the software installed in the system and of the configurations of the Operative System for each user. Analysing the register could provide useful information about the behaviour of the malware and its functionality.

Malware often uses the registry for persistence or configuration data. Once inserted inside the registry a malware can run automatically at every start-up of the system. The registry is split into the following five root keys:

- HKEY_LOCAL_MACHINE (HKLM) Stores settings that are global to the local machine
- HKEY_CURRENT_USER (HKCU) Stores settings specific to the current user
- HKEY_CLASSES_ROOT Stores information defining types
- HKEY_CURRENT_CONFIG Stores settings about the current hardware configuration, specifically differences between the current and the standard configuration
- HKEY_USERS Defines settings for the default user, new users, and current users

The two most commonly used root keys are HKLM and HKCU. These keys are commonly referred to by their abbreviations. (Sikorski, et al., 2012).

The software RegShot (Buecher) allows taking a registry snapshot that can be stored and later compared with another one. Taking a snapshot prior and after the execution of a malware can underline the changes executed inside the register.

Another software for monitoring registry was RegMon, now discontinued and integrated inside Process Monitor.

3.2.10. File System and Process Monitor

Process Monitor is a tool for Windows that shows Registry, file system and process/thread activity for an advanced monitoring. It shows event properties such session IDs and user names, full thread stack, reliable process information and much more. Process Monitor is a core utility for troubleshooting and malware hunting for its uniquely powerful features.

The *Process Explorer* display consists of two sub-windows. The first window shows a list of the active processes. The second can shows the handles that the process selected in the first window has opened. Otherwise, if *Process Explorer* is in DLL mode it will shows the DLLs and memory-mapped files that the process has loaded.

The unique capabilities of *Process Explorer* make it useful for tracking handle and DLL errors, and provide insight into the way Windows and applications work.

The software are both integrated inside the Sysinternals Suite (Microsoft).

3.2.11. Network Monitor

Network monitoring is a key sector for understanding the behaviour of a malware. The presence or not of network activity can rapidly identify the kind of malware that has been analysed.

The information that can be gained is the amount of traffic that is generated and the types of traffic. It is possible to see what is the payload of the messages exchanged, and if it is encrypted or not.

This kind of network analysis can be performed with a software like Wireshark, and it is not even needed that this software is installed on the infected machine. Wireshark can monitor all the traffic exchanged inside a network if it can physically reach the packets.

Network monitor can be performed also on the infected machine to see which are the processes that are opening new malicious connections and on which port. This kind of analysis is performed also with process monitor, or other specific software.

3.2.12. Virtual Machine

A Virtual Machine is a software useful for dynamic analysis. This software allows the creation of a virtualized physical machine inside the host system where it is installed. This virtual machine can be seen as another separate entity from the OS of the host machine. On each virtual machine can be deployed an independent OS. Since everything is virtualized it is easy to make a memory snapshot of the entire system. A snapshot can be restored in a successive period of time to have back the status of the virtual machine in that moment. Those snapshots can be incremental which creates an history of the points in time that can be recovered.

This is very useful in Malware Analysis were the malware creates irreversible damage or modification to a machine. Recovering a not infected snapshot allows a more easy execution of the dynamic analysis.

The drawback of using a virtual machine for malware analysis is that malware have been implemented anti-virtual machine techniques which can detect if the malware is running on a virtual machine and change its behaviour. Those techniques hinder the works of the security researchers.

The most famous software that allows the creation of Virtual Machines is VMWare Workstation (VMware).

4. The Banking Trojan Zeus

4.1. Introduction

Zeus or Zbot is a banking trojan, it was created to steal information such as banking details, login credentials and other sensitive information from the infected computer and send them back to the author of the attack.

Zbot is mainly focused on stealing bank related information, since these are the most profitable data in the short period, but it can also steal any credential that is considered useful.

In 2007, the first version of Zeus was detected. Since then, almost every year a minor update version of the original malware, Zeus v1.0, has been released such as v1.1, v1.2, etc... In 2010, the first major update known as Zeus v2 was released while in 2011, the source code of the Zeus v2.0.8.9 was leaked allowing the development of numerous forks over the years (fig.1) (Wontok Safe Central).

This malware creates a network of infected machines. Each machine is part of the Botnet and its owner is not aware of the critical situation, since the trojan runs silently in the background of the infected computer.

Once infected the zombie machine will automatically send the stolen information to a C&C server that gather all the data and is controlled by the owner of the botnet. The owner of the botnet can also issue commands to control all the zombie machines simultaneously, and can update the list of the website that have to be monitored to steal the data from the bot.

Zeus was initially sold on underground forums in a ready-to-use kit. At the time, the cost of the kit reached also several thousands of dollars. In order to avoid unauthorized copies of the kit, some initial versions of the kit were sold with a hardware license, so that only the purchaser could run it on his computer and build the

executable, this was the first time that this kind of security technique was applied to a malware. (Stevens, et al., 2010)

In the following sections, Zeus evolution since 2007 and its main features are analysed.

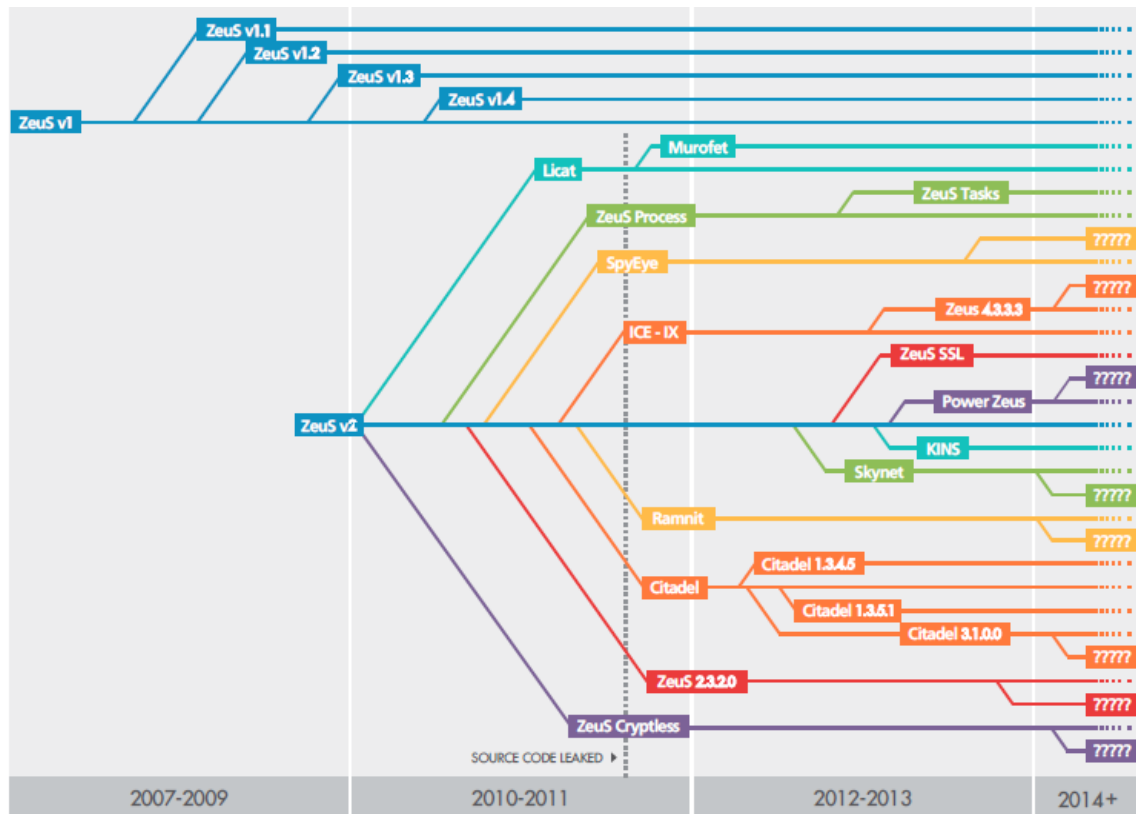


Figure 1 - Zeus timeline

4.2. History

During the years, Zeus has constantly evolved, from its first public detection in 2007 to the most recent versions.

Zeus was probably originally created in 2006 by a Russian developer, known as *Slavik*. Since then, many developers have tried to create software that could be in competition with Zeus.

The first one of this kind was SpyEye in 2009. The biggest merit of SpyEye was having a much lower price than Zeus. Moreover, its creator, the malware developer

Gribodemon, gave a specific function to the program, “kill Zeus”. This function had the objective of removing Zeus trojan from the infected system and infects it with SpyEye. This could also be considered a marketing feature.

The war with SpyEye lasted for a couple of years, during which Zeus maintained always the leadership of the sector. In October 2010, Zeus author announced his retirement on an underground forum, and his will of giving for free the source code of the software to the creator of SpyEye,

The announce of his retirement was actually a trick, since Slavik was actually developing a new version of Zeus and he deceived the abandon of the malware scene aiming to target the interest of the police to another element (Maurits, 2015). In addition to the source code, Slavik gave the ownership of all the kit customers to *Gribodemon* and put him in charge of the customers support.

Once got the source code of Zeus, *Gribodemon* claimed also the realization of a new powerful malware that was supposed to be a combination of both SpyEye and Zeus, but this was never released (Krebs, 2010).

In the year following these events, a major episode happened in Zeus history. In march 2011, someone started selling on underground forums the complete source code of the latest version of Zeus toolkit (Kruse, 2011). Rapidly, the source code was available for everyone on the internet, for free.

The free availability of a source code that was used to be sold for many thousands of dollars raised the interest of many people, from the least experts to the most skilled malware developers.

The leak of the source code kicked off the creation of many Zeus forks, and many malware have been inspired by those source codes.

Since then, many versions of the malware have been developed but some of the most important and famous Zeus forks are ICE-IX, Citadel, ZeusVM/KINS and GameOver Zeus.

Reconstructing the Zeus history after the leak is much more complex due to the quick spread of new versions and related developers.

In the meanwhile, Slavik developed a different version of Zeus that did not rely on C&C architecture. This version was known as Murofet/Licat. In September 2011, this Zeus variant morphed into peer-to-peer Zeus or also called GameOver Zeus (GOZ). The name of this morphed version was due to the fact that in one of the first version was found a link to a C&C drop zone called *gameover2.php* (Sandee, 2015).

Starting from this point, there has been a double Zeus related development, one based on the classical C&C architecture and one based on a distributed P2P architecture.

As Botnet started to be a serious treat, in March 2012, Microsoft launched an operation that disrupted a lot of Botnet based on Zeus/ICE-IX/SpyEye but this operation had no effects on the distributed entity of the botnet based on GameOver.

Differently from all the previous versions, GameOver Zeus was not sold in kit but it was exclusively used by one crime gang, leaded by Slavik.

In May 2014, another operation which specifically targeted GameOver Zeus was carried out, Operation Tovar. It was a conjunct operation which involved many different actors, from the FBI and the U.S. Department of Justice to the Europol and many security companies and universities.

At the end of the operation GameOver Zeus Botnet was disrupted, until then this botnet has made an estimated damage of 100M \$. (FBI, 2014)

FBI confirmed that Zeus was originally created from the Russian developer Evgeniy Mikhailovich Bogachev, known as Slavik. In February 2015, FBI put him in the first position of the most wanted cybercrime list of criminals with a bounty of 3M\$ for is capture.

In parallel with the development of P2P Zeus, also each one of the C&C based forks evolved during the years. In particular, KINS v1 source code was leaked in October

2013 and caused the creation of a newer version, KINS v2. This version was created in 2014 and it is currently one of the most recent versions of a Zeus-based trojan.

Recently, in June 2015, the Builder and the control panel of KINS v2.0.0.0 were leaked on internet, this event gave the opportunity to anyone of creating and using a botnet of a newer and updated version of Zeus.

The long running of Zeus over the years is principally due to the fact that it was well designed and to the leak of its source code in March 2011.

4.3. Toolkit

The Zeus toolkit is composed of several components. A scheme of all the elements is reported in figure 2.

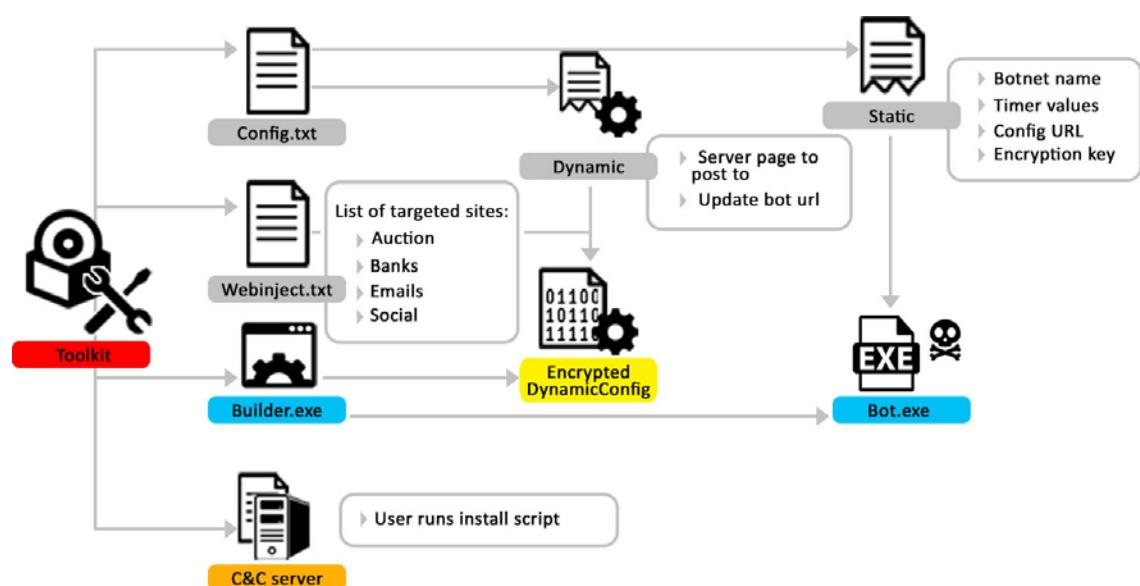


Figure 2 - Toolkit scheme

4.3.1. Config.txt

The file *config.txt* is the configuration file of the trojan (fig.3). It contains two parts the *StaticConfig* and the *DynamicConfig*.

The *StaticConfig* is read from the Builder and is embedded inside the binary of the malware. It contains the name of the botnet, the URL of the C&C server to download

the *Encrypted_DynamicConfig* and a key to do the encryption. It encloses also other fields like a backup URL if the server is not responding and timing options for the connection with the C&C server.

The *DynamicConfig* is used when the Builder needs to create the *Encrypted_DynamicConfig*, which is a different operation from the creation of the executable. It contains two URLs to the C&C server, the first is for the download of the latest version of the malware executable, and the second is a link to the drop zone of the stolen data.

The most important field in the *DynamicConfig* is the *file_webinjects* entry, which is the location where is placed the webinject file. It is essential for the creation of the *Encrypted_DynamicConfig*. It contains also other parameters irrelevant for the discussion.

```
;Version:      2.0.0.0

entry "StaticConfig"
  botnet "donato_botnet1"
  timer_session 1 1
  url_config "http://192.168.133.130/prova/test/config.jpg"
  url_reserve_config "http://192.168.133.130/prova/test/config.jpg"
  ;remove_certs 1
  ;disable_tcpserver 0
  encryption_key "donato"
end

entry "DynamicConfig"
  url_loader "http://192.168.133.130/prova/test/bot.exe"
  ;url_module_vnc "http://192.168.133.130/prova/test/mod_vnc.bin"
  ;url_module_spam "http://192.168.133.130/prova/test/mod_spm.bin"
  url_server "http://192.168.133.130/prova/Panel/gate.php"
  file_webinjects "webinjects.txt"
  entry "AdvancedConfigs"
```

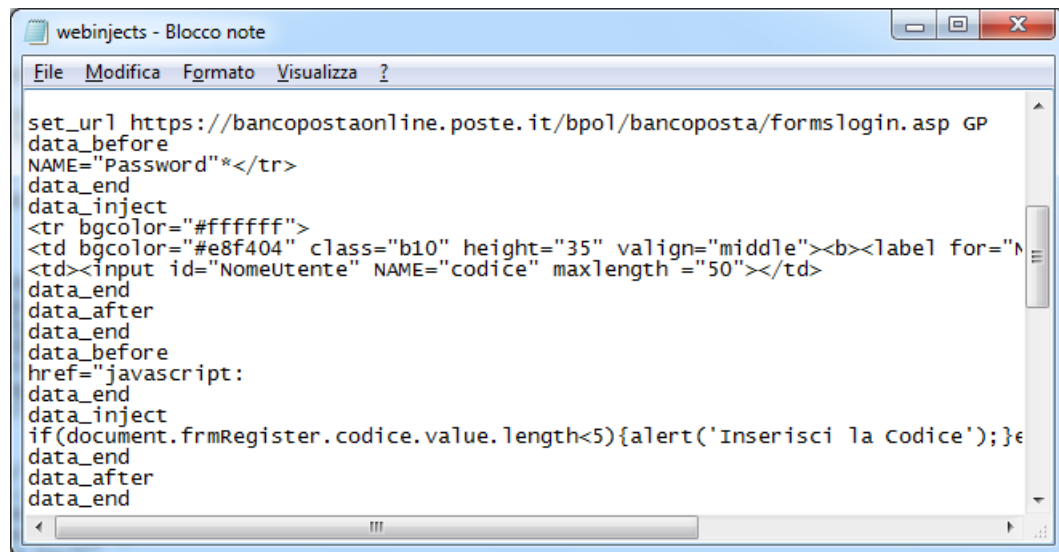
Figure 3 - Config.txt

4.3.2. WebInjects.txt

The *webinjects.txt* is an external file which contains the HTML code (fig.4). This is the core of the *Encrypted_DynamicConfig* as this file contains all the rules and the website that the malware will attack. It specifies an URL for each piece of code that needs to be injected and the position of the code inside the page.

This file is completely customizable from the owner of the botnet so he can decide which website to attack.

There is a small sample in figure 4 that log username and password from the website of the Italian bank Bancoposta.it. The file can contain an almost infinite number of rules. An underground market for gaining new and updated webinjects exists.



```
webinjects - Blocco note
File Modifica Formato Visualizza ?

set_url https://bancopostaonline.poste.it/bpo1/bancoposta/formslogin.asp GP
data_before
NAME="Password"*/tr>
data_end
data_inject
<tr bgcolor="#ffffff">
<td bgcolor="#e8f404" class="b10" height="35" valign="middle"><b><label for="M
<td><input id="Nomeutente" NAME="codice" maxlength="50"></td>
data_end
data_after
data_end
data_before
href="javascript:
data_end
data_inject
if(document.frmRegister.codice.value.length<5){alert('Inserisci la Codice');}e
data_end
data_after
data_end
```

Figure 4 – Webinject.txt

4.3.3. Command & Control Server

The botnet is controlled from a server, it has principally the role of sending the *Encrypted_DynamicConfig* and gathering all the stolen information from the bot.

To accomplish those works, the C&C has a control panel installed which is written in PHP and with a MySQL database to store the data.

The control panel is composed of two pages, the *cp.php* and *gate.php*.

The *cp.php* (fig. 5) is the page used from the owner of the botnet to check the status of the botnet, to issue commands and to read the results of the data stealing, while the *gate.php* is the page where the Bots connect to upload the information.

Figure 5 – Builder Control Panel

4.3.4. The Builder

The most important component is the builder. The builder is a windows executable program (fig. 6), with a user friendly interface and which supports a double language English and Russian. The builder has the main role of creating the malware executable of the Zeus trojan and the encrypted dynamic configuration file.

The builder takes in input the *config.txt* that contains the characteristics of that particular trojan to create the executable.

The other main function of the builder is the creation of the *Encrypted_DynamicConfig*, which is an operation that always takes in input the *config.txt* but is parallel to the creation of the executable and could be done just to update the *Encrypted_DynamicConfig*. The encryption is done with the Key provided in the *StaticConfig*.

The builder has also the function of checking if a computer is infected providing the decryption key, and a routine to delete the trojan from the infected computer.

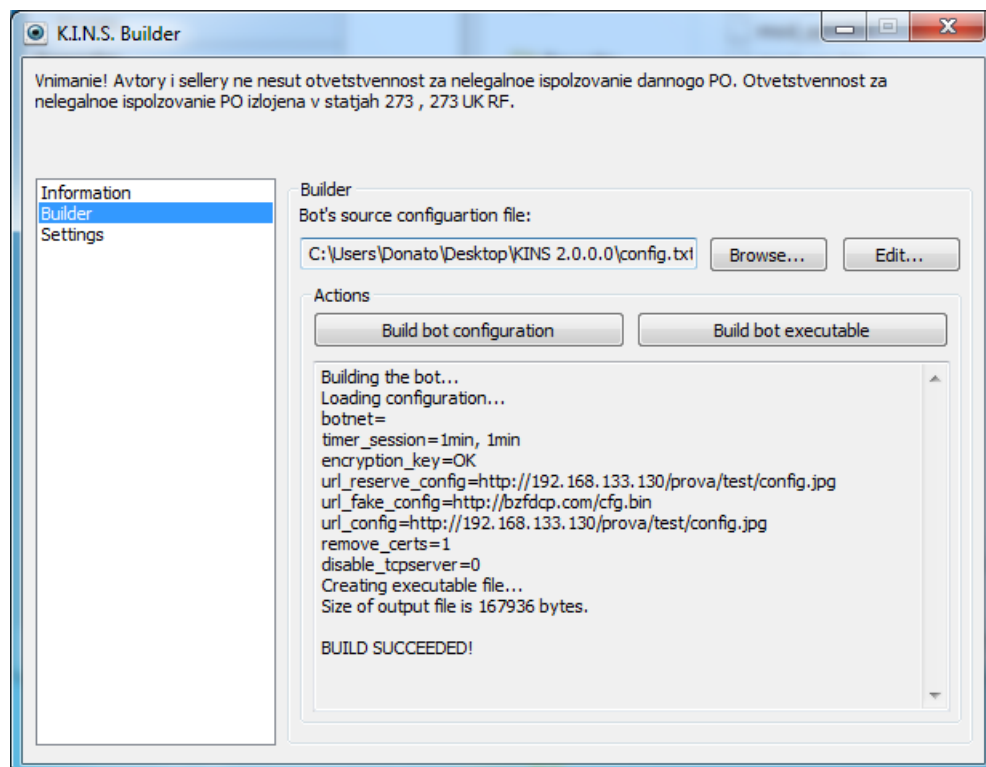


Figure 6 – ZeusVM Builder

4.3.5. The Executable

The binary file is built by the builder and it is the trojan that will be executed on the victim machine. Each version of the executable created from the same builder is identical to the others in terms of functionalities but is different for the *StaticConfig* embedded encrypted inside it.

4.4. How Zeus works

A brief overview of how most of the Zeus versions work is reported in this section.

Since its creation in 2006, Zeus was designed to mainly work on Windows XP operating systems. During its evolution, the support for more updated OS has been implemented, like Windows Vista and Windows 7.

Generally, Zeus needs to be executed on the system to infect it. After the execution, even if the installation process does not succeed the binary executed is automatically deleted from the system.

During the installation process, the trojan creates a copy of itself into a specific folder and creates a persistence key in the register to be executed at every reboot of the system. Then this new copy of the malware is executed and takes care of injecting itself inside the running process of the system. During the process, it also downloads the DynamicConfig to gather the updated information and the C&C information. At this point Zeus is ready to steal the data, the main features that has every Zeus trojan is the Man-in-the-browser. This technique uses the browser and injects piece of html code inside web pages, only for the website present inside the DynamicConfig. Through this technique, it is possible to create new form that could foolish the user to insert more personal data and track the data inserted. The websites that are most frequently involved with the injection are bank websites or other websites useful for social engineering.

Once harvested, the data are sent to the C&C URL specified in the DynamicConfig and are collected inside a database for future utilization.

5. Reverse Engineering of ZeusVM

This study focuses on the version of Zeus known as KINS/ZeusVM v2.0.0.0. This research examines this particular version of the trojan Zeus, since it is one of the most recent versions currently available and because the toolkit, containing the builder and the control panel, was leaked and made accessible online to everyone in June 2015 (Malware Must Die, 2015).

The study was carried out through a Static Analysis of the malware executable and the investigation of its behaviour.

According to these premises, the analyses performed in this research were based on the previous study about ZeusVM. In particular, the recent study of Dennis Schwarz, an employee of the security company Arbor Networks who published in August 2015 a document with technical details of ZeusVM. Although this document was written by a security expert and was published recently, it is not an official and peer reviewed research paper, so the aim of this thesis is also to verify if what is written corresponds to reality.

All the analyses were conducted on a Windows 7 machine with an Intel i7 processor and 10GB of RAM.

5.1. Case Study Environment

In the creation of the case study environment, 4 steps can be identified as reported below:

1. *Creation of the virtual machines*
2. *Installation of the Control Panel*
3. *Creation of the Malware*
4. *Tools setup*

5.1.1. Creation of the Virtual Machines

Initially, a software that allows the creation of several instances of virtual machines was installed, the VMware workstation. Each virtual machine created by the software was logically separated from the others. Three virtual machines were created in total (fig. 7). The first virtual machine was a Windows 7 machine used to run the server. Then, the second and the third virtual machines with Windows XP sp3 were created. All the machines were connected through a virtual network provided by VMware.

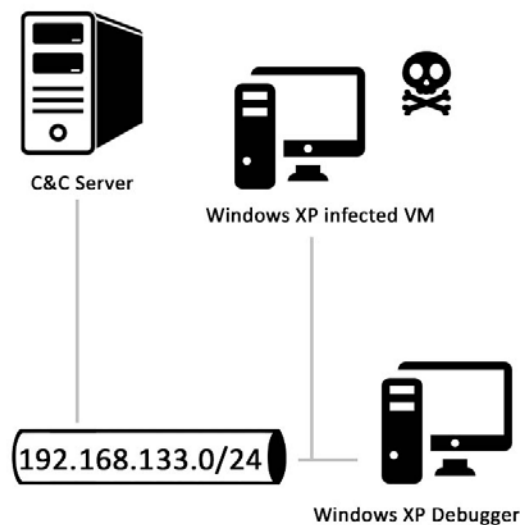


Figure 7 – Environment

5.1.2. Installation of the ZeusVM Control Panel

The Windows 7 machine was selected to be the C&C server of the Botnet. In order to turn it into the C&C server of the Botnet, a software that creates the web server with at least PHP and MySQL was needed. The software selected for this purpose was XAMPP for Windows (Apache Friends), a famous cross platform tools that allows the creation of a webserver with few easy steps.

Once installed the Apache web server and the MySQL compatible server, the system is ready to install the Zeus control panel.

The control panel is provided in the same package with the leaked builder of the malware. All the components of the control panel are placed on the web server, and they are installed in the system with the *install.php* page which is located into an install folder. To run the installation the access to the *.php* page hosted on the server with a browser is needed. In order to complete the installation, it is required to complete the prompted form from the *php* pages to setup the database and a password for the control panel.

Once installed the control panel is ready to use through the access to the page *cp.php*.

5.1.3. Creation of the ZeusVM trojan

After the installation of the control panel, the following step is the creation of the malware through the builder.

1. *Config.txt*

Firstly, the *config.txt* was modified to be suitable for this installation. Inside the *StaticConfig*, the entries *url_config*, *url_reserve_config* with the proper URL to the dynamic config located in the C&C server and the *encryption_key* with a string were compiled.

Inside the *DynamicConfig*, the *url_loader* and *url_server* were modified the first with a URL to a copy of the malware hosted on the C&C and the second with a URL to the *gate.php* page installed previously in the server. All the other fields except the *webinjects* were commented with the character “;” since they were optional and not necessary.

2. *Webinjects.txt*

The *Webinjects* file was modified to do some basics functions and mostly to test the effects of the injection. In particular, a function that steals the information inserted in the website of the bank BancoPosta was added and a popup with an incremental number to track the updates was created.

3. Builder

After this step, the builder was executed to create the malware executable and the encrypted configuration file. The malware binary was created through the function Build bot executable providing in input the *config.txt*.

Then is executed the function Build bot configuration that prompted a window to select an image .jpg to inject inside the *Encrypted_DynamicConfig*.

Once created, both the files were placed on the server in the paths specified by the static and dynamic configuration.

5.1.4. Tools Setup

On both the XP SP3 machines, software to do static and dynamic analysis were installed. The disassembler and debugger IDA Pro the SysInternalSuite with the software Process Monitor, Process Explorer and others. Moreover have been installed an hex-editor WinHex (X-Ways), and the browser Firefox.

At this point, a static IP to every machine was assigned so that each machine could always be reached with the same address. Once assigned the addresses, the malware executable was downloaded from the C&C to the Windows XP virtual machine selected for the infection, renamed to XP_TEST. The other virtual machine was renamed XP_CLEAN.

This second XP machine was not originally planned and was a later addition to the network. Its purpose was the Remote Debugging of a process inside the infected machine. The IDA Pro installation folder of XP_CLEAN was shared in the network in order to allow the remote debugging.

5.2. Analysis

The analysis of the malware was performed following some steps, according to the Static and Dynamic types of the analysis.

In figure 8, an overview of the aspects researched during the analysis knowing the works done before the starting is illustrated.

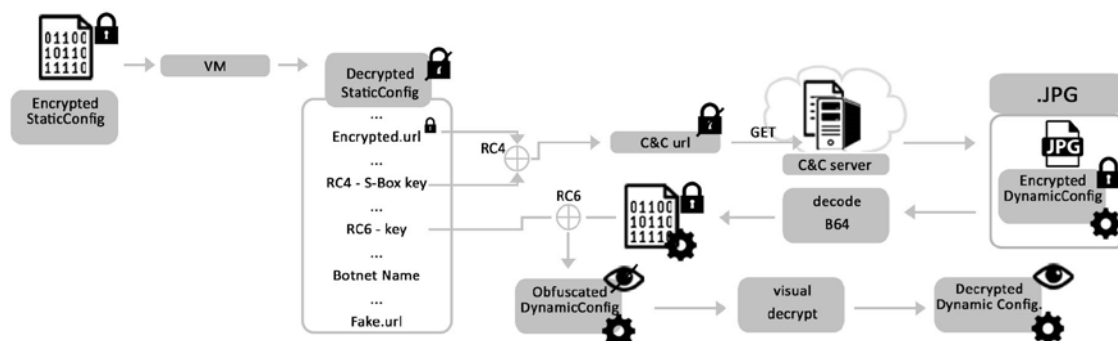


Figure 8 – ZeusVM decryption overview

5.2.1. Malware testing: Basic Static Analysis

Once created the malware, it was tested with various antiviruses.

The malware was copied from the virtual machine into another one that was security protected from the antivirus Avira Antivir. This procedure immediately raised a virus alert.

Proceeding the analysis with the antivirus, the file was uploaded to the website VirusTotal (VirusTotal) that performs a much more complete analysis between 54 different antiviruses. In this case, the malware was detected from the majority of the antiviruses, with a detection rate of 47/54 from all the antiviruses (fig. 9).

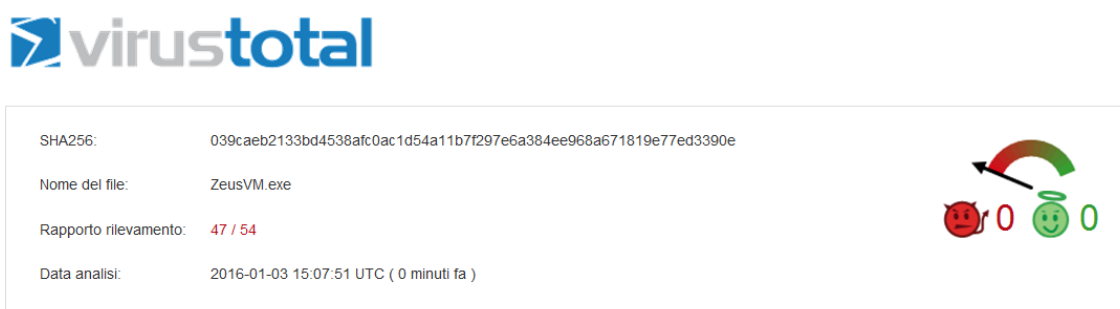


Figure 9 – VirusTotal analysis

These results confirmed the malicious behaviour of the created trojan but also that it could be easily detected in systems provided with an updated antivirus.

Secondly, an analysis of the malware executable with PEiD (aldeid) was executed. This test (fig. 10) revealed that the malware was not packed and the outcome was double-checked using also another software, PE.Explorer (HeavenTools).

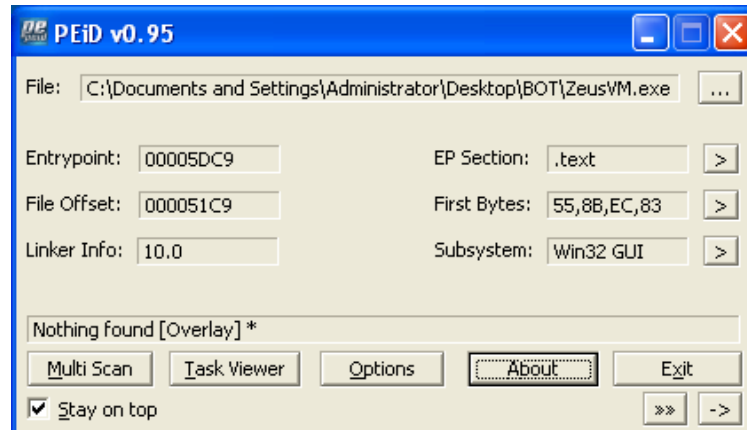


Figure 10 – PEiD analysis

These outcomes revealed that the builder does not have an automatic packer implemented inside it, and that the packing of the malware is an optional step left to each creator which can be carried out using other software, also because a common packing technique for all the sample would be more identifiable.

Proceeding the basic Static Analysis with a Header inspector like PView (Radburn) (fig.11) revealed the structure of the malware, which is divided in 4 section, .text, .rdata, .data, .reloc.

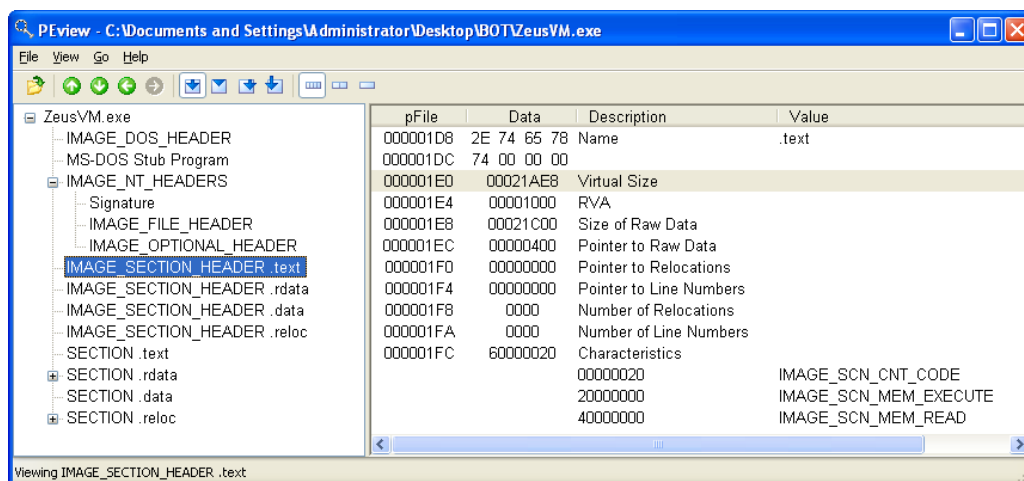


Figure 11 – PEview analysis

At this point, the malware was analysed using the software Bintext (McAfee) (fig. 12) to research some valuable strings but it did not provide relevant information. This was due to the fact that the malware was encrypted or obfuscated.

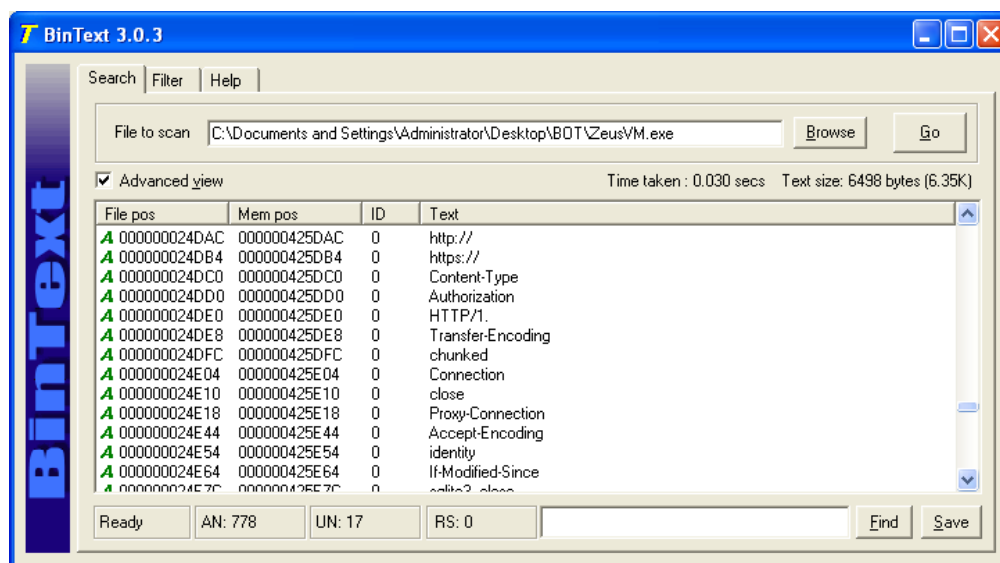


Figure 12 – BinText analysis

Although, most of the strings were encrypted, some interesting values were detected. Some strings in plain that revealed the use of *http/https* and the execution of a *.bat* file, but neither the URL or the *.bat* file were completely in clear (fig. 12).

5.2.2. Advanced Dynamic Analysis

The Advanced Static Analysis of the malware was performed, the selected tool used in this process is IDA Pro. Once opened with IDA Pro the malware is analysed automatically and the sectors inside it and the entry point of the malware are identified. The malware is automatically divided from IDA Pro in 4 sections: *.text*, *.idata*, *.rdata*, *.data*.

Inside the *.idata*, it is possible to see the static import that the malware does. It imports *kernel32.dll* and *user32.dll* and also the imported functions from those libraries. The *.text* section contains all the assembly code operation of the malware, while in *.rdata* and *.data* there are the data of the malware, with the first section.

The malware is disassembled and an automatic meaningless name with few small exceptions is assigned to all the functions inside it. In order to move inside the assembly code of the malware, the software provides a link for call and jump with a click on the address of the function.

5.2.3. Static Analysis of the Virtual Machine

Starting from this point instead of doing a blind search of information, the available data about Zeus were used as starting point. The published documents that were available for ZeusVM were analysed to identify its most peculiar feature, the virtual machine.

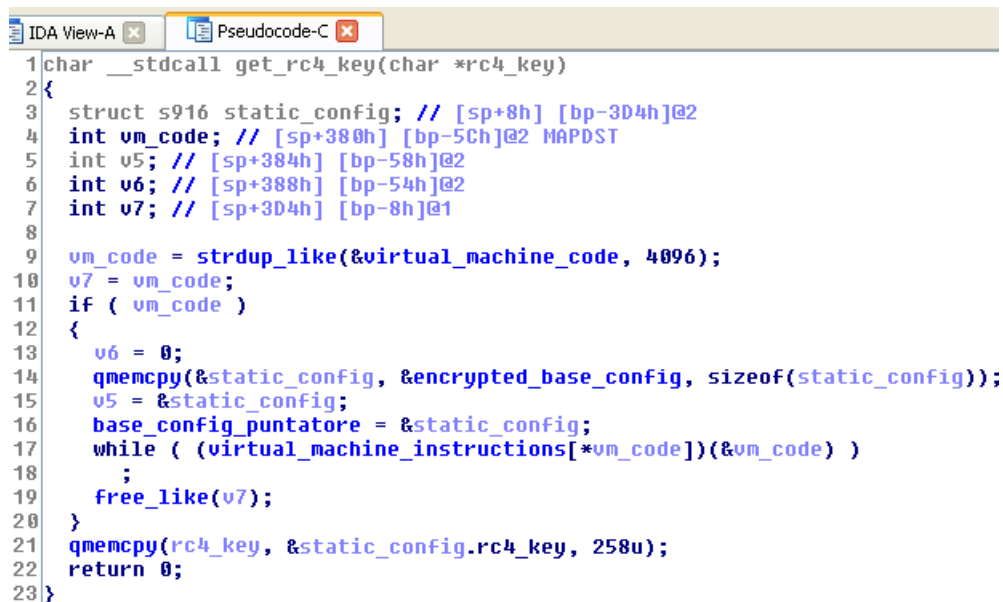
As stated by Schwarz (2015) and previously by Bijl (2013) the virtual machine is identifiable through a MOV operation of 0x1000 bytes. It is possible to search a series of byte in hexadecimal representation. The search gave many results, the second MOV 0x1000 it has a structure corresponding to the researched function.

A comparison of the structure of the function with that identified in the aforementioned analysis revealed many similarities. The variables of that function were renamed according to what was known about them. Moreover in Schwarz (2015), a pseudocode representation of the code of the virtual machine was present,

through a reconstruction of it using the decompiler plugin it resulted very similar to the pseudocode representation even if not identical.

At this point, the inspection of the portions of code called from the Virtual machine revealed three main sections.

The first was the 4096byte loaded as the first operation of the virtual machine, the second was the encrypted code of the *StaticConfig* and the third one was a list of offsets called inside a while loop. It was interesting to notice that both the Virtual machine code and the *Encrypted_StaticConfig* were inside the *.rdata* sector of the malware, and they were one subsequent to the other.

The image shows a screenshot of the IDA View-A window with the Pseudocode-C tab selected. The pseudocode is for a function named `get_rc4_key` which takes a `char *rc4_key` as an argument. The code defines a `static_config` struct, initializes several local variables (`vm_code`, `v5`, `v6`, `v7`), and then enters a loop. Inside the loop, it copies data from `encrypted_base_config` to `static_config`, updates pointers, and iterates through `virtual_machine_instructions` using `vm_code`. The loop ends with a `free_like(v7)` call. Finally, it copies the `rc4_key` into `static_config.rc4_key` and returns 0.

```
1 char __stdcall get_rc4_key(char *rc4_key)
2 {
3     struct s916 static_config; // [sp+8h] [bp-3D4h]@2
4     int vm_code; // [sp+380h] [bp-5Ch]@2 MAPDST
5     int v5; // [sp+384h] [bp-58h]@2
6     int v6; // [sp+388h] [bp-54h]@2
7     int v7; // [sp+3D4h] [bp-8h]@1
8
9     vm_code = strdup_like(&virtual_machine_code, 4096);
10    v7 = vm_code;
11    if ( vm_code )
12    {
13        v6 = 0;
14        memcpy(&static_config, &encrypted_base_config, sizeof(static_config));
15        v5 = &static_config;
16        base_config_puntatore = &static_config;
17        while ( (virtual_machine_instructions[*vm_code])(&vm_code) )
18            ;
19        free_like(v7);
20    }
21    memcpy(rc4_key, &static_config.rc4_key, 258u);
22    return 0;
23 }
```

Figure 13 – Virtual Machine

Through an analysis of the while loop (fig. 13), it was possible to see that the functions points to a memory area composed of 69 memory offsets of 4 bytes inside the *.data* sector. Each offset points to a different function inside the *.text* section. All the functions were grouped together in a compact memory space even if they were not in the same order as in the *.data* section.

Summarising, the Virtual Machine resulted composed of four parts. The first part is composed from the bytecode of the virtual machine and is always 4096 bytes. The

second part is the data over which the virtual machine has to work, which is the *Encrypted_StaticConfig*. The third part is a loop which scans the opcodes of the virtual machine and call the right handler while the fourth part are the operations of the virtual machine that are called from the handler, those operations are almost all basic operation ADD, SUB, MOVE, etc.

Once identified this first part of ZeusVM malware it was possible to observe that the first function identified was not equal to that showed from Schwarz (2015), neither from Bijl (2013). Going back to the research of the MOV 0x1000 bytes of the initialization of the Virtual machine, it arose that there are 14 different functions, which have the same identical initialization phase. Analysing the assembly and the decompiled code of those functions, the function responsible for the decryption of the RC4 (Rivest, et al., 2014) key embedded in the *StaticConfig* was found (Schwarz, 2015).

Once identified the function, it was named *get_rc4_key* and all the other functions were renamed with a name from *VM1* to *VM13* according to the order they were found in the *.text* section of the code.

The *Encrypted_StaticConfig* as presumable has fixed size for each sample created from the same builder.

In figure 13 it possible to see the function *sizeof(StaticConfig)* that denotes the variable size of the configuration. In order to have this kind of visualization, that is an automatic function from IDA Pro, the Struct which composes the *StaticConfig* had to be defined. By going into the Stack of the selected function, it is possible to manually select the bytes which form the Struct and assign them a name and a type. The software does not allow the creation of the Struct from a selection of bytes if the first and the last byte are undefined, that is the standard status of each byte of the Stack function. The Struct was defined corresponding to the *StaticConfig* starting from the byte pointed from the memcopy, the size was stated by the function that in this case was 888 bytes.

5.2.4. Dynamic Analysis

Once defined the Struct, the field corresponding to the RC4 key was defined as shown in figure 13. Once performed this first part of Advanced Static Analysis, the information gathered were verified through an Advanced Dynamic Analysis with IDA Pro which was carried out using the “Local Win32 debugger”. Prior to the beginning of the analysis, a snapshot of the virtual machine was taken so that it was possible to revert the analysis to a clean status of the system.

In order to do the analysis, a breakpoint was placed in the Start function, the first instruction of the malware, and another one in the `get_rc4_key`. Starting the debug, the modules loaded on the right panel, *gdi32.dll*, *kernel32.dll*, *ntdll.dll*, *user32.dll*, are the first things that appear. Running the debugger to the next breakpoint, it surprisingly never hits and the malware finishes its execution after some seconds in which log other modules in the output window and IDA Pro closes the debugging window.

5.2.5. Basic Dynamic Analysis

In order to understand some of the behaviour of the malware, a Basic Dynamic Analysis was executed. The malware was executed on a clean snapshot of the system, with Process Monitor and Process Explorer running in the background. The execution of the malware was too fast for Process Explorer to view useful information since the program can show only the living process. Instead Process Monitor logs every action performed by each executable so reading the log it was possible to see that the malware created another executable *fytoh.exe* in a folder *%AppData%\Maule* then an instance of the command prompt is opened and it is executed a bat file.

All those operations seemed to be familiar for the Zeus family analysis, the executed malware was deleted probably through the execution of the Bat file and another copy of the malware was installed in the system. Using Process Monitor, it is possible to see also many Registry activities but those are not clearly readable in this form.

Aiming to analyse the register, the software RegShot was used.

Firstly, a snapshot of a clean status of the system was created as well as a snapshot after the execution of the trojan.

As expected many differences between the two snapshots were detected. The most important one is the persistence key inserted from the malware inside the registry in HKU\ \Software\Microsoft\Windows\CurrentVersion\Run\epuz.exe: ""C:\Documents and Settings\Administrator\Dati applicazioni\Ewas\epuz.exe"".

This persistence key allows the execution of the malware also after the reboot of the system.

Comparing the two executables with a software that highlights the differences of the binary, like WinDiff, the original bot.exe and the new dropped.exe resulted identical, except for a block of code at the end of the file that had a size of 496 bytes.

This behavior was detected also in Wyke (2011) for the version 1 and 2 of Zeus.

5.2.6. Dynamic Analysis of Dropped.exe

At this point, there were two executables to analyze, the dropper bot.exe and the dropped.exe. Running the debugging phase several times, it was observed that the dropped.exe has always a different name and a different folder inside the *%AppData%* path of the selected user. A copy of this file and folder was made in a safe environment to take a sample of the dropped.exe to analyze; in particular the executable is \Maule\fytoh.exe.

Then the Dynamic Analysis has been moved to the dropped executable, to see which functions were called from it.

Firstly, it was checked if the "get_rc4_key" was executed inside fytoh.exe. Starting from a clean environment, the dropped executable was placed inside its folder in *%AppData%\Maule\fytoh.exe* and has been launched the debugger with IDA Pro. The debugger had almost the same result, it did not hit the breakpoint and crashed at the end of the execution. The same anomalous behavior was detected also for the process Explorer.exe.

Aiming to understand this behavior, some tests were performed.

It resulted that running directly the dropped from a clean environment inside its folder, it launched correctly the ZeusVM trojan even if the dropper is not executed. Nevertheless, executing the dropped from a debugger like IDA Pro and setting up some breakpoints made the program crash without installing the trojan. The first thought was that there could be some anti-virtual machine techniques or anti-debugging techniques.

Since the executable was working outside the debugger and there was no problem with VMware, the problem was identified in the debugger. This was probably due to the presence of some timing check since there were accesses to the sleep function.

In addition, it was found out that the problem does not rise and the debugger does not crashes exchanging the type of breakpoints to hardware and tracing the first four functions . It was possible to trace all the program execution without a crash of the program, so there were no timing checks.

The problem with the hardware breakpoints was that they are limited to 4. The presence of a function that performs a CRC32 (Walma, 2007) check was found, it is identifiable searching the peculiar number involved in the computation of the algorithm 0xEDB88320. Since this function is called many times maybe it is not a security check for the anti-debugging.

The answer for the crash is that the insertion of a breakpoint changes a byte inside the code. The change is revealed by the CRC32 hash function check, this creates some anomalies inside the structure of the program that brings it to not working. These kinds of problems are very time consuming.

5.2.7. Dynamic Analysis of RC4 S-Box

The analysis continued with the other functions called from the bot.exe which have the initialization of the Virtual Machine, a breakpoint was placed on each function and the debugger was launched, everything in a VMware snapshot of a clean environment.

The functions executed were: VM3, VM1 and VM4 several times. Then the analysis of the operations behind the virtual machine was started beginning from the function VM1. Since each virtual machine was initialized with the same code, on the same portion of data, they could perform the same operations.

The Virtual Machine, during the initialization, copies the content of the *Encrypted_StaticConfig* inside the Stack and saves the address as a pointer in a global variable. This global variable is the same for each Virtual Machine initialization found in the VM functions.

After this phase, the Virtual Machine was executed through the while loop. At the end of the execution of the Virtual Machine, the global variable points to the *Encrypted_StaticConfig*. At this point, it was possible to analyze the data decrypted and if all the functions are equivalent, it should be possible to find the RC4 S-box key inside it. The *Decrypted_StaticConfig* was saved in hexadecimal, through the export function in the Hex-view window of IDA Pro. Aiming to verify the presence of the RC4 S-box key, a python implementation of the KSA algorithm was used to generate the S-box starting from the encryption key that was provided to the builder.

Once generated the S-box, it was compared with WinHex to see if there was a matching. A matching was found at the offset 0x15F, the S-Box generated externally, with the same seed, corresponded to the 256 bytes which were present inside the *Decrypted_StaticConfig*.

The same correspondence was found inside the *StaticConfig* Struct defined in IDA Pro, the offset 0x15F referred to the first byte of the RC4 S-boxes. This proved that even if the function involved is different from that stated by Schwarz (2015) the configuration is still decrypted in the same way and the RC4 S-box is the same.

It was interesting to find out that the RC4-Sbox was only of 256 bytes while what was loaded from the *StaticConfig* is always a series of 258 bytes, this is due to the fact that the last two bytes loaded for the decryption are the indexes "i" and "j" of the RC4-

PRNG algorithm. Those bytes although are always loaded are set to zero in every sample analyzed created from this builder.

5.2.8. Static Analysis of RC4 PRNG

Among the instructions of the Virtual Machine, the instruction that implements the RC4 algorithm, the instruction_22, was found. Inside its body, it was possible to identify the 2 main parts of the RC4 algorithm, the KSA algorithm to create the S-Box and the PRNG algorithm that performs the XOR operations (Rivest, et al., 2014). Once identified a function, with the command X of IDA Pro, it is possible to see all the references inside the malware to that function. Starting the analysis of the PRNG function, it is possible to see that it was referenced by two still known functions, VM4 and VM13.

In particular, the VM13 was analyzed since the size of the bytes used in the PRNG function corresponded to what Schwarz (2015) says about the decryption of the C&C URL.

Inside the VM13 function, two PRNG very similar decryption function were found. Both the functions decrypt 101 bytes of a different offset inside the *Encrypted_StaticConfig* through a key that is mapped to the same local variable. The image inserted inside (Schwarz, 2015) seems to refer to the second function. Running the Dynamic Analysis should confirm it.

5.2.9. Remote Debugging of Explorer.exe

Since the function VM13 was never called inside the dropper, the functions called inside the dropped.exe were analyzed, but it was not called in that executable too.

Running Process Monitor, it is possible to verify that during the execution of the malware, portions of code are injected inside Explorer.exe, the presence of an <unknown> object is visible inside the Stack Summary of the process.

At this point, an Advanced Dynamic Analysis of the process Explorer.exe was required. As the problems with the debugging of Explorer.exe are that if the process is paused,

the system becomes unusable, in order to avoid working in a frozen system, it is possible to work with a remote debugger.

The remote debugging is the procedure of debugging a process from another machine. At this stage, another instance of a Window XP to accomplish the role of remote debugger was created with VMWare.

IDA Pro provides all the necessary for the remote debugging inside its program folder. In order, to start the remote debugging, the infected machine and the debug machine need to share the folder dbgsrv of IDA Pro installed in the debug machine.

Aiming to start the remote debugging, the infected machine has to launch win32_remote.exe and once executed, the machine starts to listen incoming debug connections.

At this point, it is possible to start the remote debugging. In particular, it is needed to select the Remote Windows debugger inside IDA Pro and setup the IP of the target machine. Once completed the setup, it is possible to debug a remote application or attach to a remote process.

In order to continue the analysis, +the remote debug of the Explorer.exe process was started through the attach function.

Once started the debug, since it was a new executable, all the previous defined functions were lost. Moreover, the position of the injection was not known.

An easy way to find where the code was injected is starting a signature based search of the function of interest. A binary search of the hex code "68 00 10 00 00 68 F0 43", that corresponds to the "PUSH 1000" which is the initialization of the virtual machine, was performed.

After a while of searching through the file, the desired sign was found. Going to the location pointed by the search, a section of the process marked as Data was identified. At this point, this section was manually converted to Code with the command C of IDA

Pro and an automatic analysis. After having converted all the VM functions, a breakpoint was placed on each one.

5.2.10. C&C URL Decryption

The first hit breakpoint is inside the function VM13. Following its execution, it was possible to see that the PRNG (fig. 14) function executed is the first and not the second. Moreover, it was found that the decryption function adopted was not the RC4 S-Box as stated by Schwarz (2015) but instead it was the reverse array of the RC4 S-Box.

It was observed, that the function prior the PRNG loads the 256 bytes of the S-Box starting from the last to the first. The second PRNG function was not encountered during this phase of the debugging. In a second moment, it was discovered that it was the function that decrypts the *url_reserve_config*. In this second case, the URL is decrypted with the RC4 S-Box.

```

bug159:023E0AB0 loc_23E0AB0: ; CODE
bug159:023E0AB0 inc [ebp+var_1]
bug159:023E0AB3 movzx esi, [ebp+var_1]
bug159:023E0AB7 mov dl, [esi+eax]
bug159:023E0ABA add [ebp+var_2], dl
bug159:023E0ABD movzx ecx, [ebp+var_2]
bug159:023E0AC1 mov bl, [ecx+eax]
bug159:023E0AC4 mov [esi+eax], bl
bug159:023E0AC7 mov [ecx+eax], dl
bug159:023E0ACA movzx esi, byte ptr [esi+eax]
bug159:023E0ACE mov ecx, [ebp+arg_0]
bug159:023E0AD1 movzx edx, dl
bug159:023E0AD4 add esi, edx
bug159:023E0AD6 and esi, 0FFh
bug159:023E0ADC mov dl, [esi+eax]
bug159:023E0ADF xor [ecx+edi], dl
bug159:023E0AE2 inc edi
bug159:023E0AE3 cmp edi, [ebp+arg_4]
bug159:023E0AE6 jnb short loc_23E0AB0
bug159:023E0AE8 pop esi
bug159:023E0AEA pop ebx
bug159:023E0AEA loc_23E0AEA: ; CODE
UNKNOWN 023E0ADF: RC4_PRNGA_Decrypt+4F

```

Hex View-1

02F2F594	8C 12 0F 56 92 EA CB 3F 0B 7A A2 EF 75 37 D8 5F	î..U&0-?.z6'u7Ï
02F2F5A4	6E 11 DE 50 BF 4D CF E7 01 A9 21 C0 F3 22 D6 56	n.İP+M&þ.0!+&''iU
02F2F5B4	5E 92 12 66 0F D4 F7 52 F6 BA B4 73 D2 B0 1C 7E	^&.f.Ê.R:!!sÊ!~
02F2F5C4	E4 59 2D B5 77 B3 FB BA 16 6B 9F 19 5E A5 FD 1C	ôY-Aw!'.kM.^Nz.
02F2F5D4	68 74 74 70 3A 2F 2F 31 39 32 2E 31 36 38 2E 31	http://192.168.1
02F2F5E4	33 33 2E 31 33 30 2F 70 72 6F 76 61 2F 74 65 73	33.138/prova/tes
02F2F5F4	74 2F 63 6F 6E 66 69 67 2E 6A 70 67 00 63 6A E5	t/confirm.jpg.cj0

Figure 14 – URL Decryption

5.2.11. DynamicConfig Decryption

Continuing the analysis of the VM functions executed, it was possible to see that the VM11 function was executed. It was noticed that this function, differently from the others, after the decryption of the *StaticConfig* calls another function that loads a different offset instead of that of the RC4 S-box.

Following this function, it was found that there was a memcpy of 176 bytes inside it from the decrypted *StaticConfig*. As stated by Schwarz (2015), this is the RC6 (Rivest, et al., 1998) key expansion output of the Key Schedule Algorithm of RC6, even if the function is different from that reported in the report. The function that copies the RC6 key simply calls another function.

Analyzing the execution of this function using the debugger, it was shown that there was a while loop with a XOR operation that runs inside it.

Analyzing the Hex-view of the memory locations XORed, it was possible to see in clear text the decrypted *DynamicConfig*.

Analyzing in details the while loop it was observed that it performs the RollingXOR Algorithm, also called VisualEncrypt/Decrypt, which is the final step of the decryption of the *DynamicConfig*. The other decryption functions have to be between the acquisition of the key and the RollingXOR algorithm.

The function that executes the RC6 decryption is called before the rolling XOR and it creates the decrypted code in a memory area by executing that algorithm on a memory space that should be the *Encrypted_DynamicConfig*. To verify this, the *Encrypted_DynamicConfig* has to be analyzed.

The *Encrypted_DynamicConfig* was downloaded from the C&C server through an image file config.jpg, but it was known that inside this image there is the *Encrypted_DynamicConfig* since the file was modified through the builder to embed inside it the dynamic configuration.

Opening the config.jpg with the editor WinHex, it was possible to see clearly that there is a data field appended at the end of the file. This data field is inserted as a comment inside the .jpg. The comment is identifiable from the standard marker “FF FE” and finishes with the marker “FF D9”. The size of the comment is a 4 bytes field which is placed 10 bytes after the comment marker. Then there is the comment, and observing its ASCII representation inside WinHex it is possible to recognize that it is encoded in Base64 (Josefsson, 2006) as also stated by Schwarz (2015).

config.jpg																	
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00001E90	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001EA0	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001EB0	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001EC0	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001ED0	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001EE0	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001EF0	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001F00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001F10	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	28	A2	8A	00	(čŠ (čŠ (čŠ (čŠ
00001F20	28	A2	8A	00	FF	FE	3F	10	00	00	50	FF	70	B5	EC	08	(čŠ yb? Pypui
00001F30	00	00	39	37	6D	32	53	44	43	56	65	5A	2B	36	4B	73	97m2SDCVeZ+6Ks
00001F40	55	52	2B	32	54	4E	65	34	6C	54	47	48	4C	33	69	46	UR+2TNe4lTGHL3iF
00001F50	31	2F	72	47	4F	79	39	6D	74	45	79	72	4E	4E	73	30	1/rGOy9mtEyrNNs0
00001F60	41	52	48	67	46	6A	35	56	4E	62	42	6D	4E	47	74	57	ARHgFj5VNbBmNGtW
00001F70	69	43	49	34	4D	36	71	33	35	6D	55	57	59	62	38	6C	iCI4M6q35mUWYb8l

Figure 15 – DynamicConfig inside JPG

It is generally possible to decode the content of the comment with a tool. Moreover, many free online tools that encode directly the data in Hex exist.

The decoded comment inside the config.jpg was compared with the data buffer involved in the decryption with RC6. It was observed, that they corresponded and had the same bytecodes. Through the identification of the decryption function, it was possible to confirm that the involved key and encrypted data corresponded.

5.2.12. Traffic Analysis

Once obtained those encryption keys, an analysis of the TCP packets exchanged between the C&C and the infected machine was performed.

The packets were sniffed through the well-known software Wireshark. As previously mentioned, many messages were exchanged. This was due to the fact that during the

creation of the malware inside the *StaticConfig*, the timing options were set to 1minute. This means that every minute the Bot issues a HTTP GET request to obtain a new configuration and then perform a HTTP POST operation to send stolen data to the C&C drop zone (fig.16).

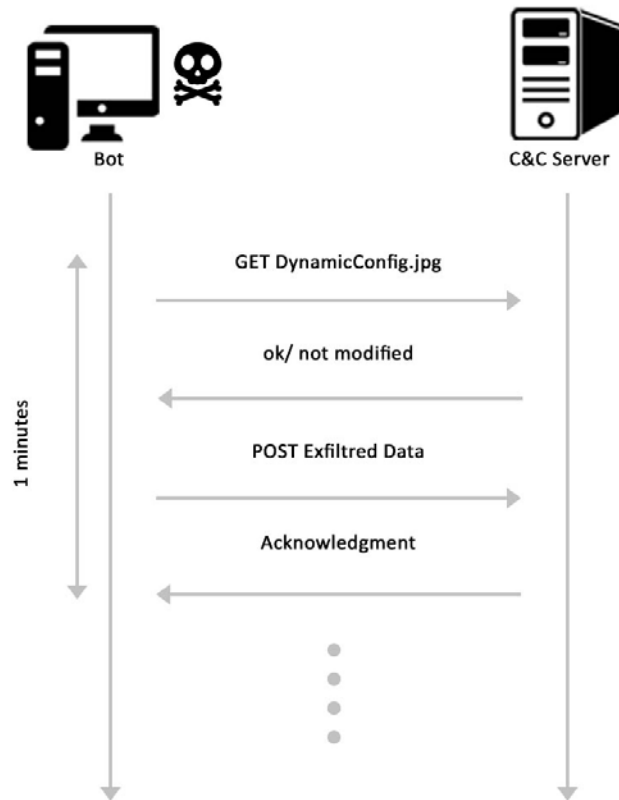


Figure 16 – Communication Bot-C&C

This standard communication is implemented by the functions VM13, VM11 and VM5. This happen when there are no new information.

As it is possible to see in figure 17, also the size of exchanged packets is always the same.

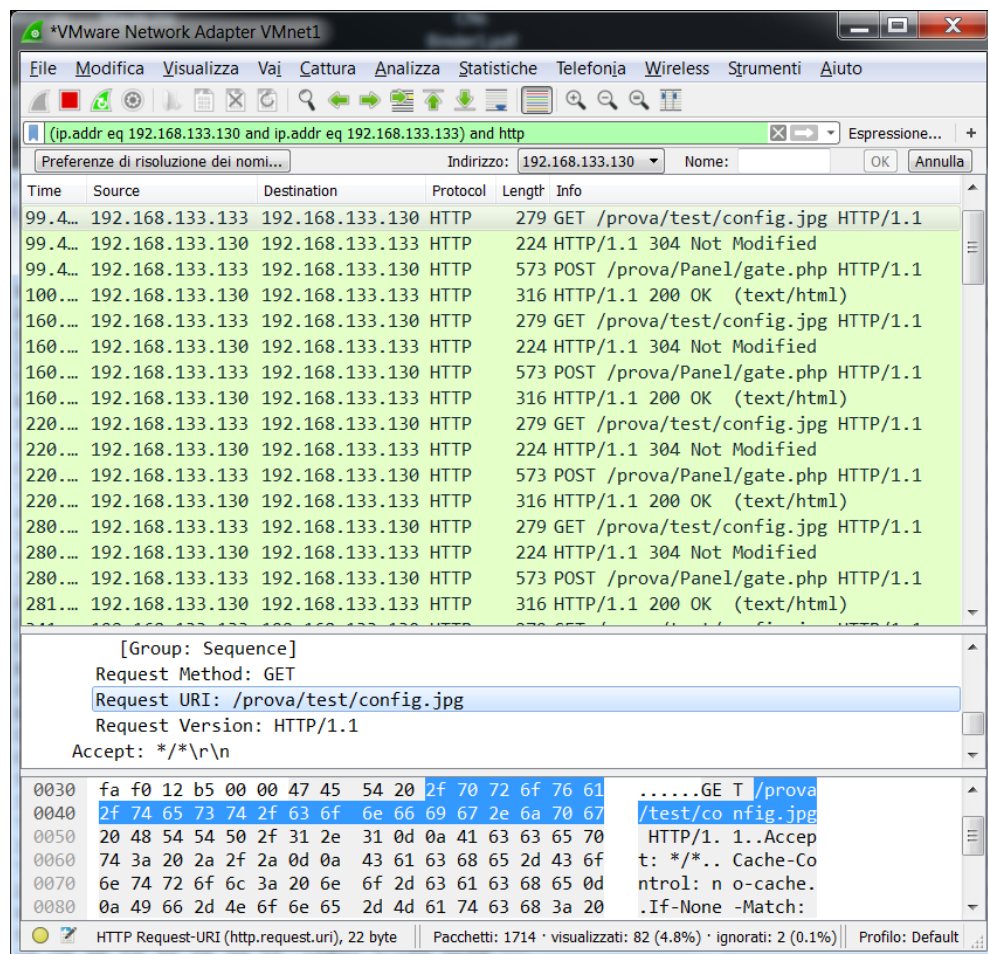


Figure 17 – Packet Exchanged Bot-C&C

5.2.13. Dynamic Analysis of communications

The behavior of the execution of the functions injected in Explorer.exe during the creation of the data to post was analyzed. A trace was placed on each VM function, and the functions which were triggered during the execution were analyzed.

In order to test the update of the configuration, a new file was created with the builder. This file was almost identical to the previous one, but with an incremented number to show the difference. By visiting a selected website, it will prompt on screen a number representing the updated configuration to check the effectiveness of the update.

The update process was monitored both on IDA Pro and Wireshark. On Wireshark, it was possible to see that the packets have been transferred and that the Server replied with a 200 OK message, so the transfer was completed. Moreover, it was possible to inspect the reassembled packet with the ASCII content of the image.

On IDA Pro, a new function was triggered during the process of the acquisition of the new config.jpg: the VM12. After the update of the configuration, in the next time period the behavior of the program was back to its normal course. By opening with a browser the webpage with the incremental number injected, the success of the update was enlightened.

A similar process was performed for the upload of stolen information. Inside the webinject.txt it was implemented a real functioning malicious webinject found online, to steal the data from the Italian bank BancoPosta.

By visiting the selected website, inserting username and password, and requesting to login, the upload process was started. Even if the data inserted in the fields were not correct for the login, they are uploaded. However, this depends from the structure of the webinject.

Probably due to the debugger, the loading of the webpage was very slow respect to its normal behavior. The browser used to do the test was Internet Explorer v6, the browser preinstalled on the system. It was possible to notice that the SSL layer was not compromised during the injection. Doing the same test with the latest version available of Firefox, v43, the pages were loaded more quickly and the webinject was executed correctly.

With Wireshark it was possible to see the new packets exchanged with the HTTP POST, and their size resulted different from the previous one. The content of the packets were saved for further analysis.

Watching the trace windows of IDA Pro, like in the previous case, it was possible to see a new function used in the execution: VM3.

5.2.14. Static Analysis of POST data

The packets extracted with Wireshark were analyzed. The content was not in clear text so it was probably encrypted or obfuscated.

Since for the DynamicConfig the RC6 algorithm was used, this was the first attempt. As decrypting the packet with the RC6 algorithm did not give any result. Then the decryption with the RC4 algorithm was tested. Performing the decryption with RC4 key inserted inside the builder, or directly with the corresponding S-Box gave some results. The data seemed to have a structure inside it, so the code should be still obfuscated. Since during the decryption of the DynamicConfig, the use of the RollingXOR algorithm or VisualEncrypt/Decrypt was discovered, this algorithm was the first one to be tested for the deobfuscation.

In order to perform those operations, a python script was created, mainly to apply the RC4 decryption and the RollingXOR algorithm of the data inserted in the packet. The result of the deobfuscation was a readable text (fig. 18).

In the first rows of the documents, there was a data structure but the fields were readable. What kind of structure was used was not further investigated.

In the corps of the document it is possible to read the username and password used for the login and all the other information.

```
1 T    ?D          3    SYN?    DC1
2 >    '          CAN    CAN    XP_TEST_7875768F7A0B1440    DCS
3 >    '          EOT    EOT    STX    ES'    ACK    ACK    STXETX(    GS
7 <    '          NAK    NAK    XP_TEST\AdministratorSOH    "
9 <    '          ZVT    ZVT    https://bancopostaonline.poste.it/bpol/bancoposta/Logon.fcc
10 User input:https://bancopostaonline.poste.it/bpol/bancoposta/formslogin.aspxdonatodonato
11 Request:
12
13 POST /bpol/bancoposta/Logon.fcc HTTP/1.1
14 Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash
15 Referer: https://bancopostaonline.poste.it/bpol/bancoposta/formslogin.aspx
16 Accept-Language: it
17 Content-Type: application/x-www-form-urlencoded
18 Accept-Encoding: gzip, deflate
19
20 USER=donato&Password=donato&dep=version%253D%2526pm%255Ffpua%253Dmozilla%252F4%252E0%252C
```

Figure 18 – Decrypted POST data

5.2.15. Multiple Malware Samples Analysis

Once the behavior of the malware was understood, other samples were created to gain new information from the possible differences.

Firstly, a new sample with an identical *config.txt* was created, and the two files were compared with a tool that analyzes the binaries, WinDiff. Three sections of different code were found, one is at the beginning and the other two are at the end. The first modified section is the Virtual Machine code section and the *Encrypted_StaticConfig*, but neither the content of the operations nor their structure were modified. The other two sections were not been analyzed.

Series of samples were created to identify other sections of the *StaticConfig* that had not been analyzed.

Through the comparison of the *Decrypted_StaticConfig* of each new executable sample with that one used the previous analysis, some new elements were discovered.

The first field modified in the *StaticConfig*, was the *botnet_name*, the input string in this field can be maximum 20characters. After running the decryption through a Dynamic Analysis of the involved sample, it was found out that the *botnet_name* was a field readable in clear after the decryption.

Previously this field was left blank in the config. In my sample, the offset of this field was 112 (fig. 19). A research was done to see which functions called this particular offset. The only function identified was VM5, which is involved in the process of uploading stolen data.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000D0	48	70	B4	36	61	87	AF	39	99	C2	F6	25	DD	3F	40	8C	Hp'6a+9mÂo%Y?@E
00000E0	A2	4A	BA	FC	A5	8E	31	6D	D7	2A	99	D3	69	8B	B9	3D	çJ°ü#Ž1m×**óí< !=
00000F0	8C	E4	7B	AD	2B	CB	FD	27	1F	F8	DB	54	05	EF	F7	AB	Æa{-+Éý' øÛT i÷«
0000100	98	2C	6C	C1	20	07	2E	4D	16	E6	51	18	6A	9F	46	9E	;11Á .M æQ jŸFž
0000110	86	C5	64	00	6F	00	6E	00	61	00	74	00	6F	00	5F	00	tÂd o n a t o
0000120	62	00	6F	00	74	00	6E	00	65	00	74	00	31	00	00	00	b o t n e t 1
0000130	CA	4D	C1	1E	CD	9D	07	E8	D8	81	8A	11	86	3B	56	05	ÊMÁ í èø š †;V
0000140	E5	AB	96	D0	20	68	5A	D6	33	00	22	00	DF	DB	6F	C6	â«-Ð hzÖ3 " ÑÛoÆ
0000150	77	76	B3	FB	00	00	00	00	C9	AD	05	A6	00	00	00	00	wv*û É- !»ÄDd
0000160	D4	D0	4B	8E	94	70	B8	15	54	96	E1	DF	00	00	00	00	ÔÐKž"p, T-ás8'S-

Figure 19 – Decrypted StaticConfig

The second field analyzed was the timing options, and repeating the same process allowed to identify them at the offsets 0x148 and 0x14A. They were contiguous and had a size of 4 bytes. The values were expressed in hexadecimal, and in this case (fig. 19), they corresponded to the decimal values 51, and 34. In comparison with the *config.txt* file, the values were swapped in their position. Analyzing the functions, those values were called only inside the VM10 function.

Continuing the analysis, the next fields were the URLs configurations. Since the main URL of the C&C server had already been analyzed, only the second one was analyzed.

In the *config.txt* taken as a sample, those values were identical, so it was harder to understand the differences. Changing the second URL, it has been observed that the maximum allowed size in the input of the *config.txt* is 100chars. Decrypting the *StaticConfig* enlightened a difference at the field 3D, the URL was not in clear as in the previous version. The field 3D was used in the VM13 function, through the analysis of the point where it was executed, it was possible to see that, similarly to the first URL, it was decrypted through the RC4 S-Box key, which was not reversed in this case. At this point, it was clear that Schwarz (2015) refer to this URL when it talks about its decryption but usually this field is not used so it could be left blank or it could be used to point to a wrong address with the aim of foolishing the security analysts. The main URL address is the first that should be analyzed.

Continuing the analysis it was found out the presence of the *remove_certs* and *disable_tcpserver* contiguous at the offset 0x38. Those field were used in the functions VM6 and VM7 but they were not analyzed.

The last sample created is also the most different, since the encryption key was changed. The executable maintained the same structure as in the previous cases, and nothing seemed different from the base case sample.

By analyzing the decrypted *StaticConfig* it was possible to see the differences. Since the configurations are equal and only the encryption keys was changed, the different sections involved in the differences were relative to the decryption key. It was possible

to see three portions of code, the first containing both the encrypted URLs, and the other two sections containing the RC4 S-box and the RC6 key expansion output. All the other sections of the *StaticConfig* were the same so they were not related to the encryption key.

5.3. Summary

The execution of the malware was analyzed in its three phases and the functions related to the Virtual Machine were monitored.

During the execution of the Bot.exe or what it could be called the “installation phase”, the VM related functions were VM1, VM3 and VM4, all involved with the use of the RC4 S-box key.

By executing only the dropper in a clean environment, and using the hardware tracing, the functions used during the injection phase were VM1, VM2, VM3 and VM4. As in the previous case, the only element extracted from the *StaticConfig* was the RC4 S-box.

During the execution inside the injected process Explorer.exe, the functions used during the analysis were VM3, VM4, VM5, VM10, VM11, VM12 and VM13.

As it can be seen in figure 20, there is a clear separation between the functions used during the process.

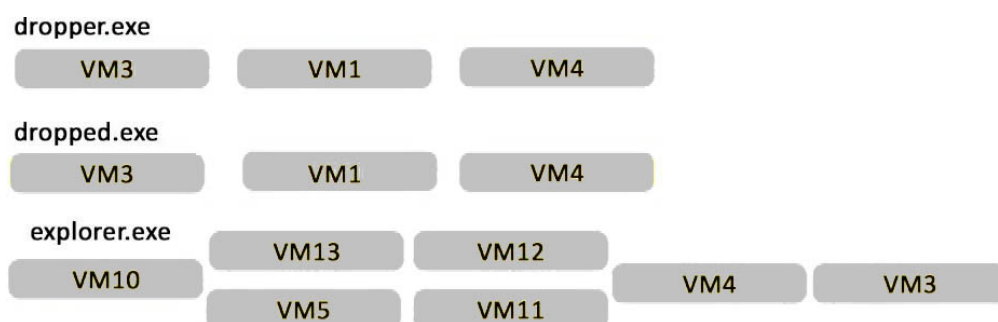


Figure 20 – VM functions during execution

Some of the analyzed functions were never executed during the observation of the system. This could be due to the fact that they are generally used only in specific situations not triggered during all the examined executions.

Moreover, it is possible to state that the functions VM6 and VM7 extract from the *StaticConfig* the not analyzed parameters *remove_certs* and *disable_tcpserver*, while the functions VM8 and VM9 extract a field at the offset 0x154 which is always 0 which could be an optional hidden parameter of the configuration. The other function never encountered is VM0 or *get_rc4_key*, this should only extract the RC4 key, and maybe it is just not used.

The main phases of the execution of the trojan are depicted in figure 21.

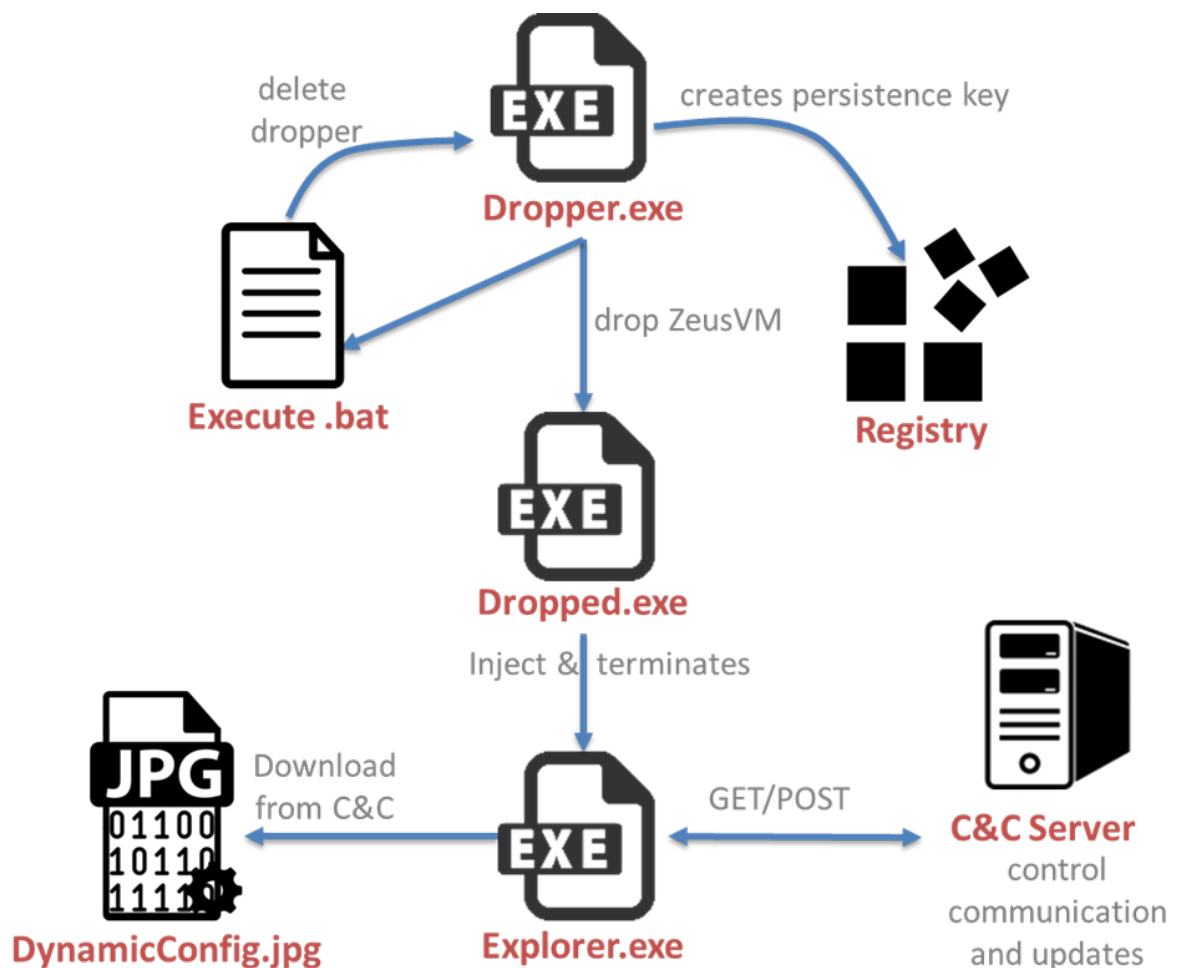


Figure 21 – ZeusVM execution

The dropper runs into the system, creates the persistence key and creates the dropped. Then launches a .bat to delete the dropper and executes the dropped. The dropped does an injection of itself inside the system process Explorer.exe and then exits. Inside the injected code the malware downloads the C&C configuration and uploads stolen data to the drop zone. To steal data, also the browsers are infected with malicious code to run the webinject. This aspect has not been analyzed.

In figure 22, there is a summary of the aspects discovered during all the analyses, with a merge of the information known from the previous works.

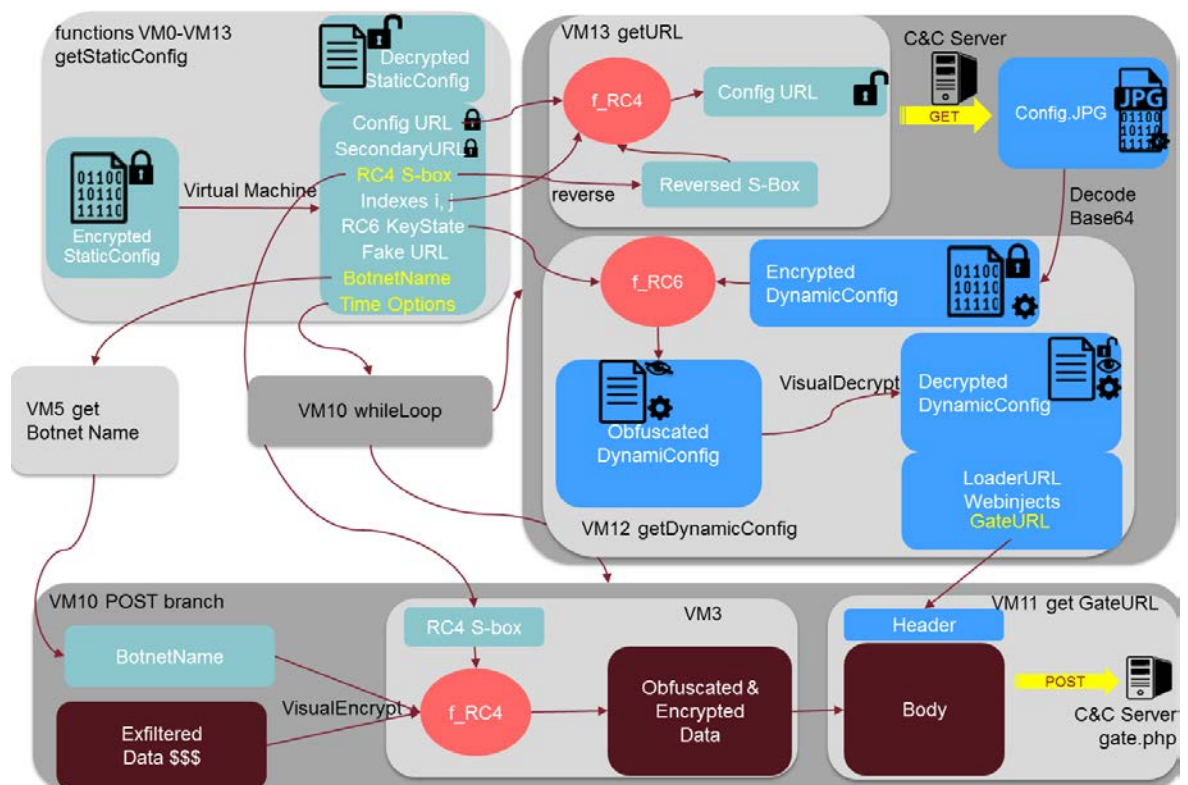


Figure 22 – ZeusVM

5.3.1. Missing pieces

The differences among the instructions of the Virtual Machine were not confirmed through the analysis. No differences were found among the samples. This could be due to the fact that only samples generated from the same builder were used.

Differences among each operation of the Virtual Machine exist, since in the comparison with Schwarz (2015) there is a different “sign”, but those differences could be only between different versions of the malware, or different compiled versions of the builder. Since only one precompiled version of the builder was available, these aspects are still unclear.

The UCL compression or the bintext structure of the decrypted DynamicConfig have not been investigated

The URL decryption stated in Schwarz (2015) is relative only to the decryption of the reserve URL and it does not work for the primary URL of the C&C. This difference in the decryption is not mentioned.

6. Conclusions

The Zeus family is very rich of samples, K.I.N.S/ZeusVM 2.0.0.0 is one of the most interesting new additions to this malware family. This trojan took its name from its most unconventional peculiarity, the Virtual Machine.

In this thesis, the structure of this Virtual Machine has been analysed. The polymorphic nature of this trojan has been confirmed, due to the fact that the decryption process realized through this Virtual Machine is always different.

The functions related to the use of the Virtual Machine have been analysed. Those functions are related to the *StaticConfig*, this has been deeply analysed to understand its fields and which role they had during the execution of the malware, with respect to the use of the Virtual Machine. In particular, it was found out an unknown scheme to decrypt the C&C URL which is encrypted and embedded inside the malware during its creation. In addition, the *DynamicConfig* has been analysed, together with the phases of its decryption through the use of the Virtual Machine. The traffic between the infected system and the Command & Control server has been analysed and its encryption scheme based on RC4 and RollingXOR algorithm has been understood.

All those results have been obtained through the application of the Reverse Engineering, in particular through the use of the Static and Dynamic Analysis. Those methods combined proved to be the best process to understand an unknown malware. In particular, the Dynamic Analysis is a required process to understand and decrypt the malware that encrypt themselves and their traffic.

6.1. Future Works

For future works, the reverse engineering of this malware could be enhanced and be performed more in depth, starting from the knowledge acquired. Other aspects of the malware could be analysed to point out new information about the behaviour of the ZeusVM trojan.

In particular, the main aspect that has not been analysed in this thesis and that could be of great interest for the future developments of web browsers is the Man-in-the-browser technique. This is due to the fact that it is still a valid technique to alter the content of a web page, and also the most popular updated browsers are affected by such flaw. The researcher could investigate also the process of the injection, to fully understand where and how the injection can be performed to running processes by this malware. This has not been investigated since it is a known technique, but every malware has its own different peculiarities.

This work could be continued in the study of other Botnets created from other trojans similar to ZeusVM, or Zeus. One of this trojan could be the trojan ZBerp, to enlighten the differences between those versions.

7. References

A.S.L Exeinfo PE [Online]. - <http://exeinfo.atwebpages.com/>.

Agarwal Shiv Kumar and Shrivastava Vishal BASIC: Brief Analytical Survey on Metamorphic Code [Journal] // International Journal of Advanced Research in Computer and Communication Engineering. - September 2013. - 9 : Vol. 2.

aldeid PEiD [Online]. - <https://www.aldeid.com/wiki/PEiD>.

Apache Friends XAMPP [Online]. - <https://www.apachefriends.org/>.

Bijl Joost Analysis of the KINS malware [Online] // Fox-IT. - 2013. - <http://blog.fox-it.com/2013/07/25/analysis-of-the-kins-malware/>.

Buecher M. regshot [Online] // sourceforge. - <http://sourceforge.net/projects/regshot/>.

Chikofsky E.J. and Cross J.H. Reverse engineering and design recovery: A taxonomy [Journal] // Software, IEEE. - 1990. - 1 : Vol. 7.

Combs Gerald [Online] // Wireshark. - <https://www.wireshark.org/>.

Damodaran A. Combining Dynamic and Static Analysis for Malware Detection [Report] / Master's Projects ; San Jose State University. - 2015. - p. 6.

Eastlake D. and Jones P. US Secure Hash Algorithm 1 (SHA1) [Online] // The Internet Engineering Task Force. - 2001. - <https://tools.ietf.org/html/rfc3174>.

Eldad E. Reversing: Secrets of Reverse Engineering. [Book]. - Indianapolis : Wiley Publishing, Inc. 10475, 2005. - Vol. 1.

FBI GameOver Zeus Botnet Disrupted [Online] // FBI. - 2014. - <https://www.fbi.gov/news/stories/2014/june/gameover-zeus-botnet-disrupted>.

Gadhiya S. and Bhavsar K. Techniques for Malware Analysis [Article] // International Journal of Advanced Research in Computer Science and Software Engineering Research. - April 2013. - 4 : Vol. 3.

Gandotra Ekta, Bansal Divya and Sofat Sanjeev Malware Analysis and Classification: A Survey [Journal] // Journal of Information Security. - 5 2014. - pp. 56-64.

HeavenTools PE.Explorer [Online]. - <http://www.heaventools.com/overview.htm>.

Hex-Rays IDA Pro [Online]. - <https://www.hex-rays.com/products/ida/>.

Josefsson S RFC4648: The Base16, Base32, and Base64 data encodings [Online] // <https://tools.ietf.org/html/rfc4648.txt>. - 2006.

Krebs Brian SpyEye v. Zeus Rivalry Ends in Quiet Merger [Online] // Krebs on Security. - 2010. - <http://krebsonsecurity.com/2010/10/spyeye-v-zeus-rivalry-ends-in-quiet-merger/>.

Kruse Peter Complete Zeus sourcecode has been leaked to the masses [Online] // CSIS. - 05 2011. - <http://www.csis.dk/en/csis/blog/3229/>.

Malware Must Die MMD-0036-2015 - KINS (or ZeusVM) v2.0.0.0 toolkit (builder & panel source code) leaked. [Online] // Malware Must Die. - 5 7 2015. - <http://blog.malwaremustdie.org/2015/07/mmd-0036-2015-kins-or-zeusvm-v2000.html>.

Maurits Lucas A Short History of Attacks on Finance [Online] // RSA Conference. - 2015. - https://www.rsaconference.com/writable/presentations/file_upload/stu-w2-a-short-history-of-attacks-on-finance.pdf.

McAfee BinText [Online] // Intel Security. - <http://www.mcafee.com/it/downloads/free-tools/bintext.aspx>.

Microsoft Sysinternals Suite [Online]. - <https://technet.microsoft.com/en-us/sysinternals/bb842062>.

Radburn Wayne J. Utilities [Online]. - <http://wjraddburn.com/software/>.

Rivest R. The MD5 Message-Digest Algorithm [Online] // The Internet Engineering Task Force. - MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992. - <https://www.ietf.org/rfc/rfc1321.txt>.

Rivest Ronald L. [et al.] The RC6 Block Cipher [Report]. - 1998.

Rivest Ronatld L. and Schuldt Jacob C. N. Spritz a spongy RC4-like stream cipher and hash function [Report]. - 2014.

Sandee Micheal GameOver Zeus Background on the Badguys and the Backends [Conference] // Blackhat US 2015. - 2015.

Schwarz Dennis ZeusVM: Bits and Pieces [Online] // Arbor Networks. - 08 2015. - <https://asert.arbornetworks.com/zeusvm-bits-and-pieces/>.

Sikorski M. and Honig A. Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software [Book]. - [s.l.] : No Starch Press, 2012.

Singhal A. and Shlok G. Reverse Engineering [Journal] // International Journal of Computer Applications. - December 2014. - 9 : Vol. 108.

Stevens Kevin and Jackson Don Zeus Banking Trojan Report [Online] // Dell SecureWorks. - 3 2010.

VirusTotal [Online]. - <https://www.virustotal.com/>.

VMware VMware Workstation Pro [Online] // VMware. -
<http://www.vmware.com/it/products/workstation/>.

Walma M. Pipelined Cyclic Redundancy Check (CRC) Calculation [Conference] // Proceedings of 16th International Conference on Computer Communications and Networks. - 2007.

Wontok Safe Central The Evolution of Financial Malware 2007-2014 [Report].

Wyke J. What is Zeus? [Report] : Technical Paper / SophosLabs UK. - 2011.

X-Ways WinHex: Computer Forensics & Data Recovery Software, [Online] // X-Ways. -
<http://www.x-ways.net/winhex/>.

Yuschuk Oleh [Online] // OllyDbg. - <http://www.ollydbg.de/>.