September 21, 2021

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

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).

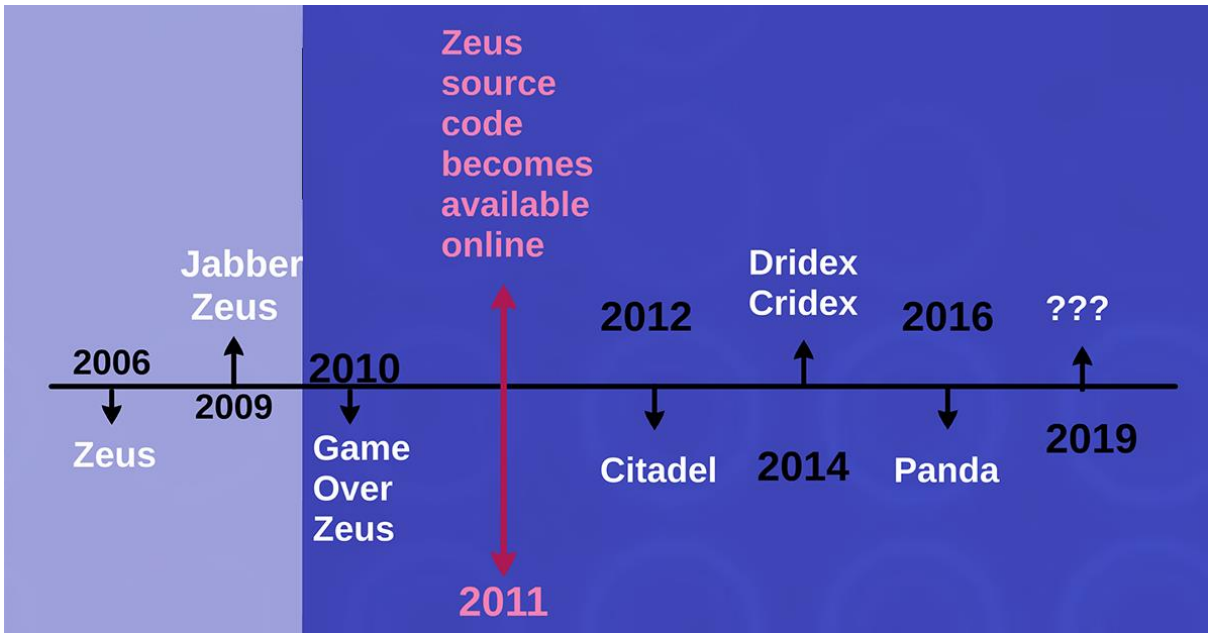


Fig. 1. Evolution of Zeus Trojan

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 webinjects item, which specifies the directory in which the webinject file is stored. It is required for the Encrypted DynamicConfig to be created.

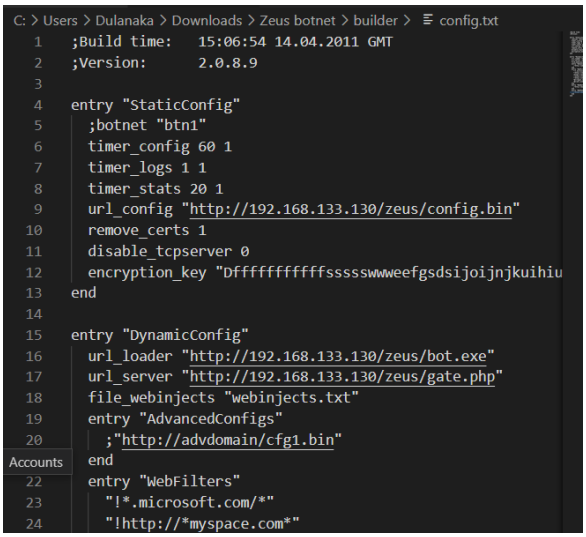


Fig. 2. Config.txt

- **WebInjects.txt**

The external file webinjects.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, webinjects.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.

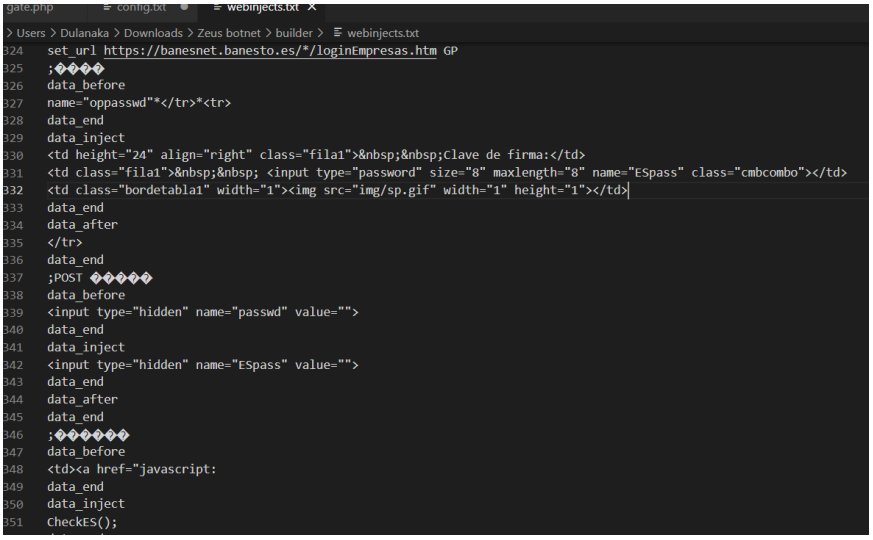


Fig. 3. Webinject.txt

- **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.

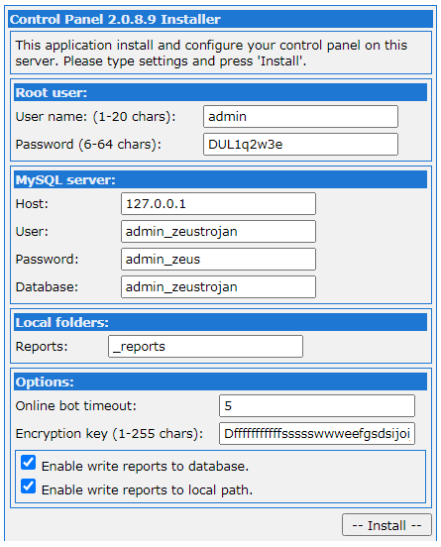


Fig. 4. C&C (Command and control panel)

- **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.).

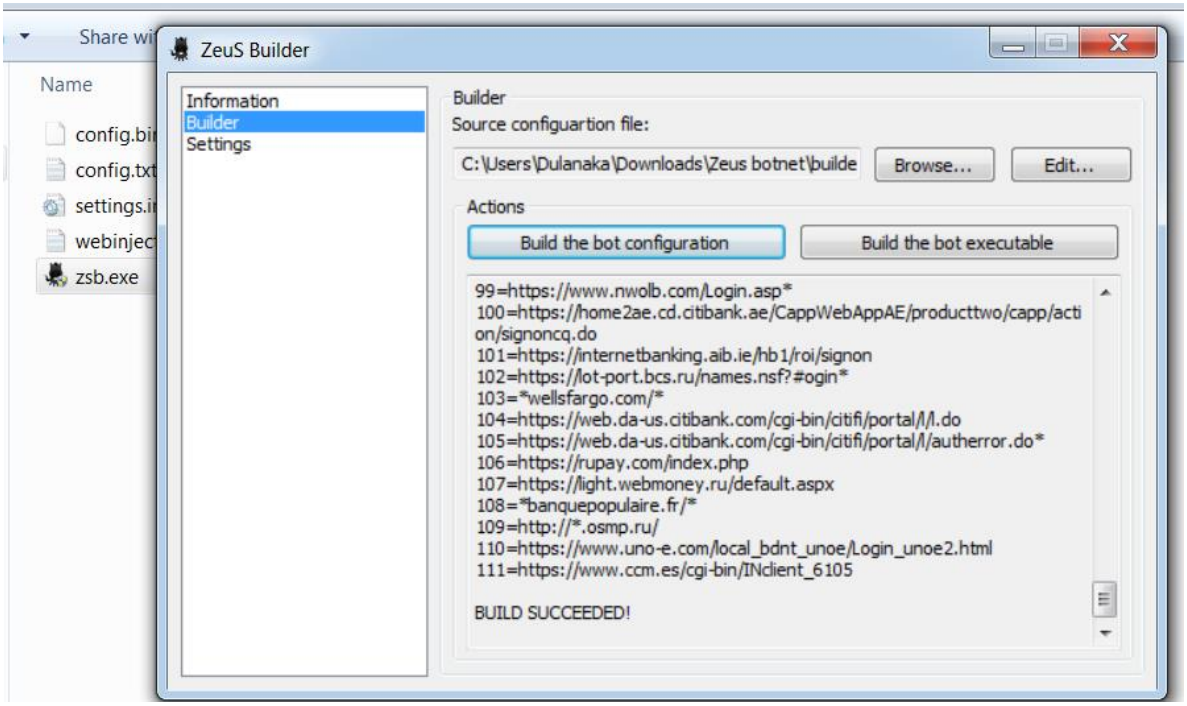
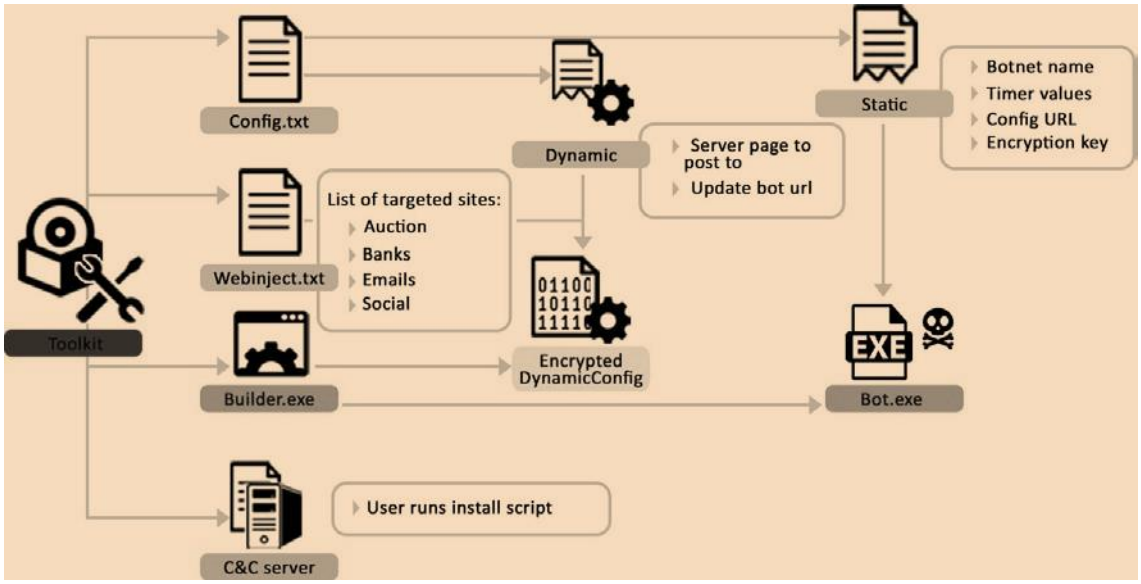
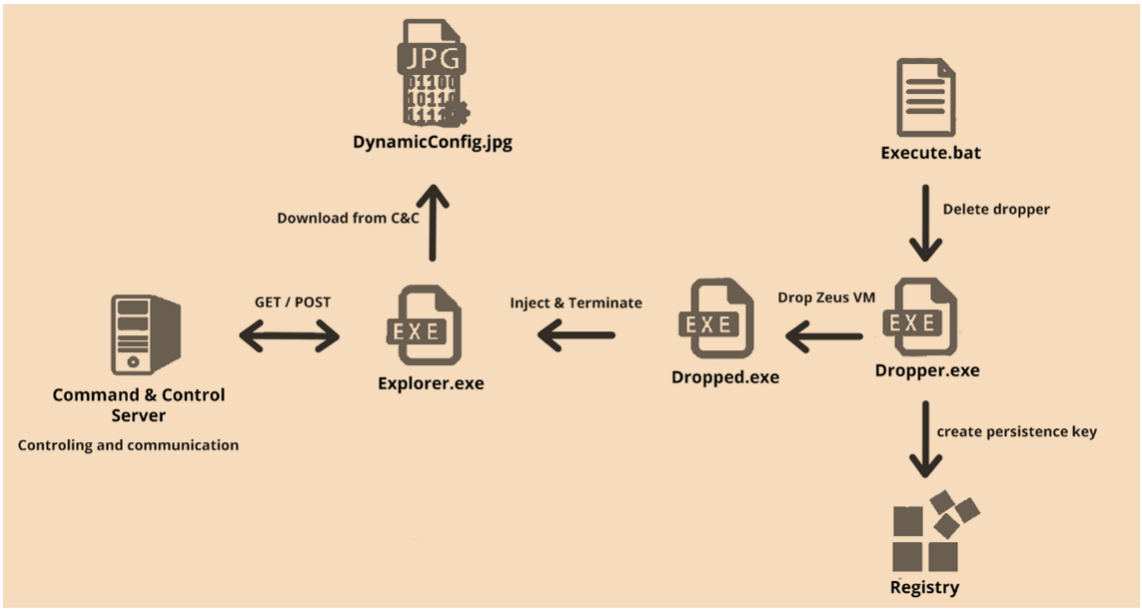


Fig. 5. Zeus Builder

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).

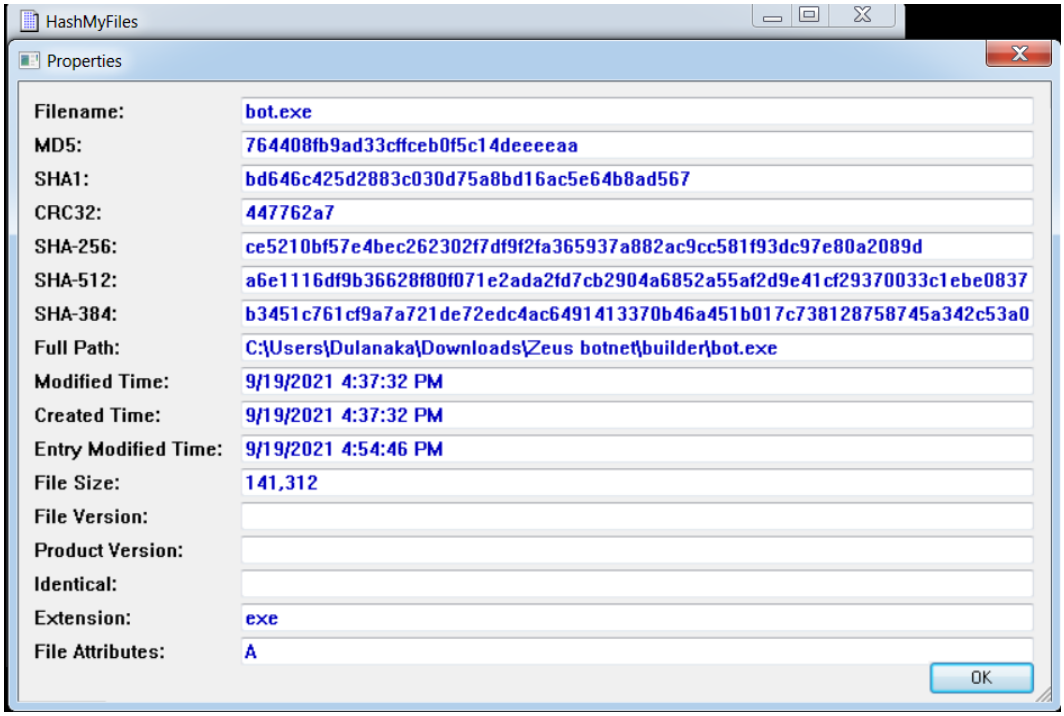


Fig. 7. Generating hash

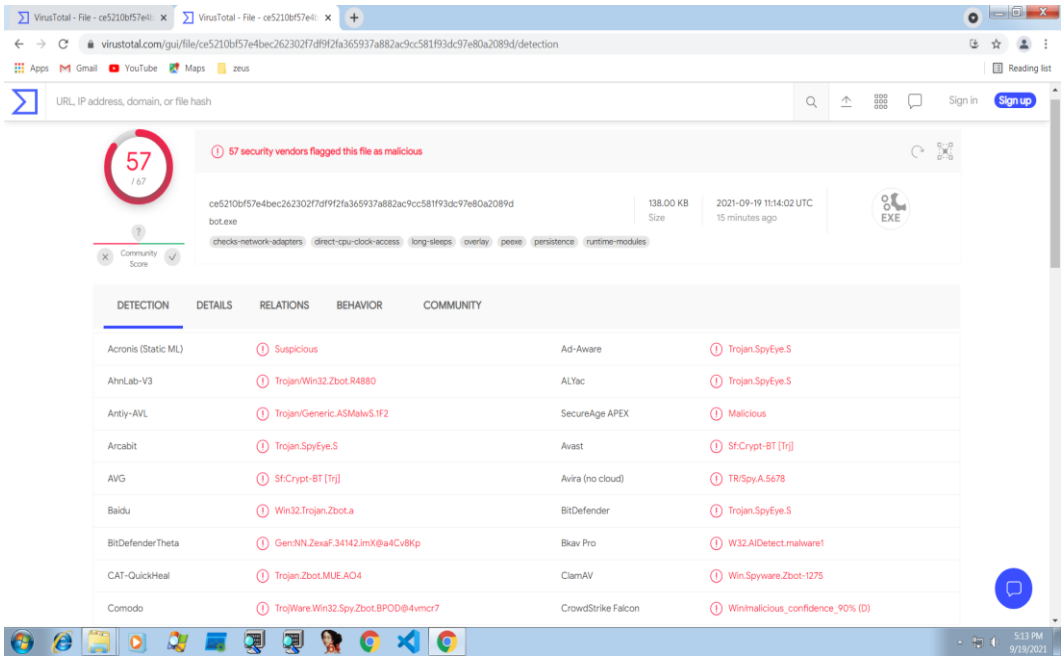


Fig. 6. VirusTotal analyse

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).

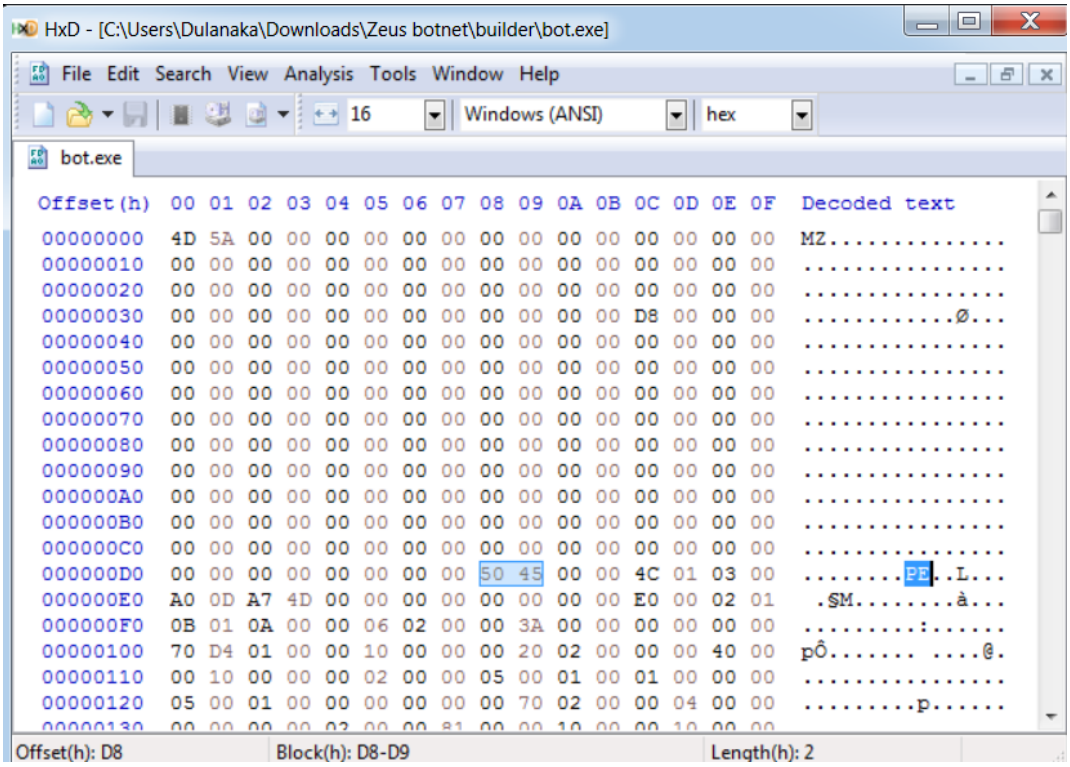


Fig. 8. HxD PE Header analyse

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).

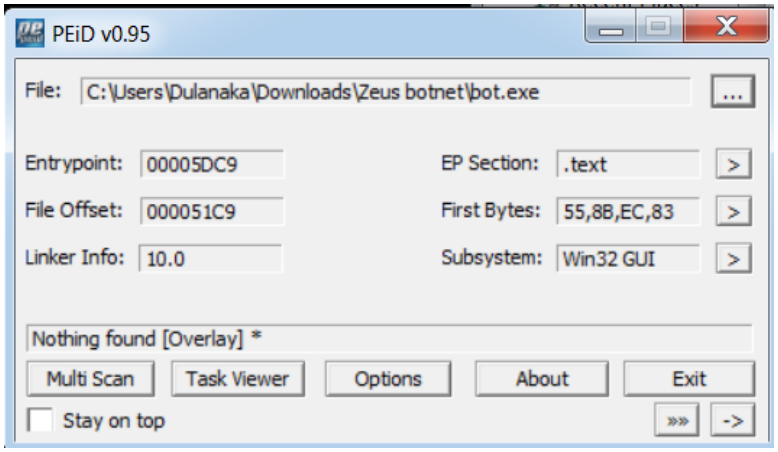


Fig. 9. PEiD Header analyse

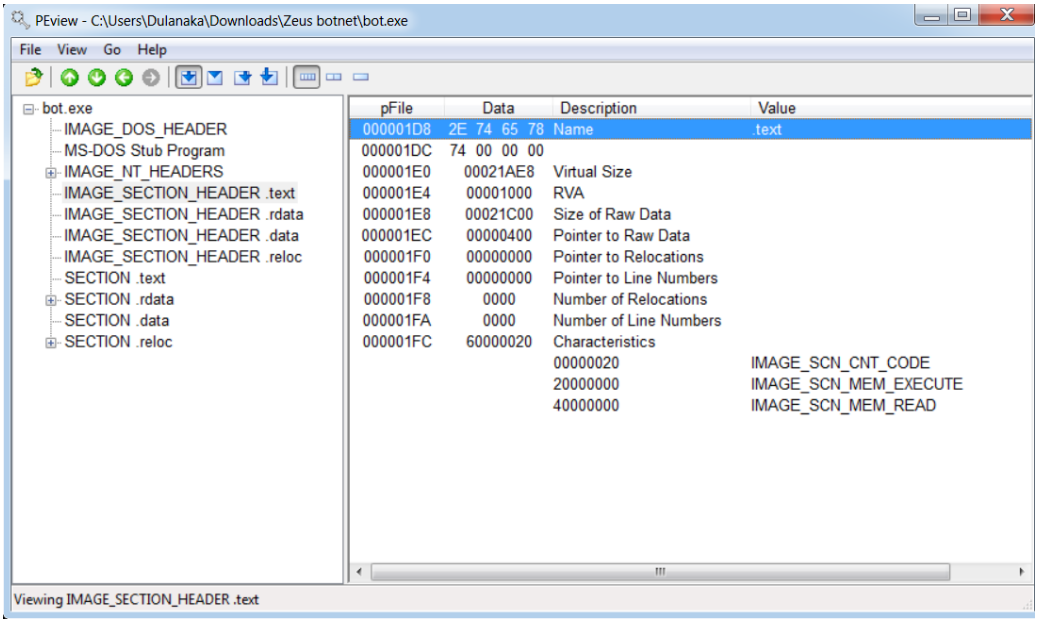


Fig. 10. PEview Header analyse

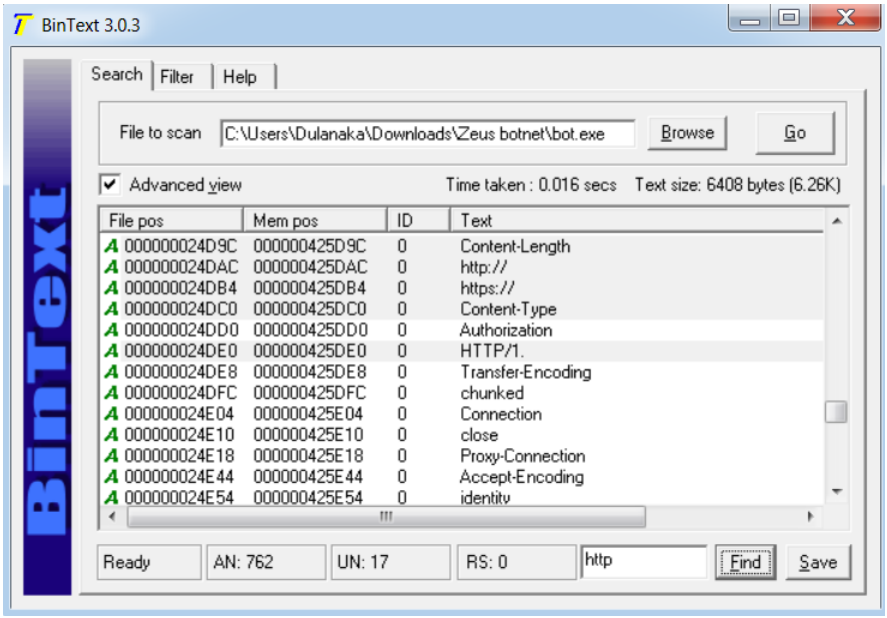


Fig. 11. BinText analyse

Rather of 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                 dd 0
.rdata:00428238 word_428238     dw 2EFh                 ; DATA XREF: .rdata:00428218↑o
.rdata:0042823A                 db 'InterlockedIncrement',0
.rdata:0042824F                 align 10h
.rdata:00428250 word_428250     dw 2CBh                 ; DATA XREF: .rdata:00428200↑o
.rdata:00428252                 db 'HeapAlloc',0
.rdata:0042825C word_42825C     dw 459h                 ; DATA XREF: .rdata:off_4281E8↑o
.rdata:0042825E                 db 'SetEvent',0
.rdata:00428267                 align 4
.rdata:00428268 word_428268     dw 24Ah                 ; DATA XREF: .rdata:004281EC↑o
.rdata:0042826A                 db 'GetProcessHeap',0
.rdata:00428279                 align 2
.rdata:0042827A word_42827A     dw 4B2h                 ; DATA XREF: .rdata:004281F0↑o
.rdata:0042827C                 db 'Sleep',0
.rdata:00428282 word_428282     dw 245h                 ; DATA XREF: .rdata:004281F4↑o
.rdata:00428284                 db 'GetProcAddress',0
.rdata:00428293                 align 4
.rdata:00428294 word_428294     dw 33Ch                 ; DATA XREF: .rdata:004281F8↑o
.rdata:00428296                 db 'LoadLibraryA',0
.rdata:004282A3                 align 4
.rdata:004282A4 word_4282A4     dw 0Eh                  ; DATA XREF: .rdata:004281FC↑o
.rdata:004282A6                 db 'AddVectoredExceptionHandler',0
.rdata:004282C2 word_4282C2     dw 215h                 ; DATA XREF: .rdata:00428214↑o
.rdata:004282C4                 db 'GetModuleHandleA',0
.rdata:004282D5                 align 2
.rdata:004282D6 word_4282D6     dw 52h                  ; DATA XREF: .rdata:00428204↑o
.rdata:004282D8                 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.

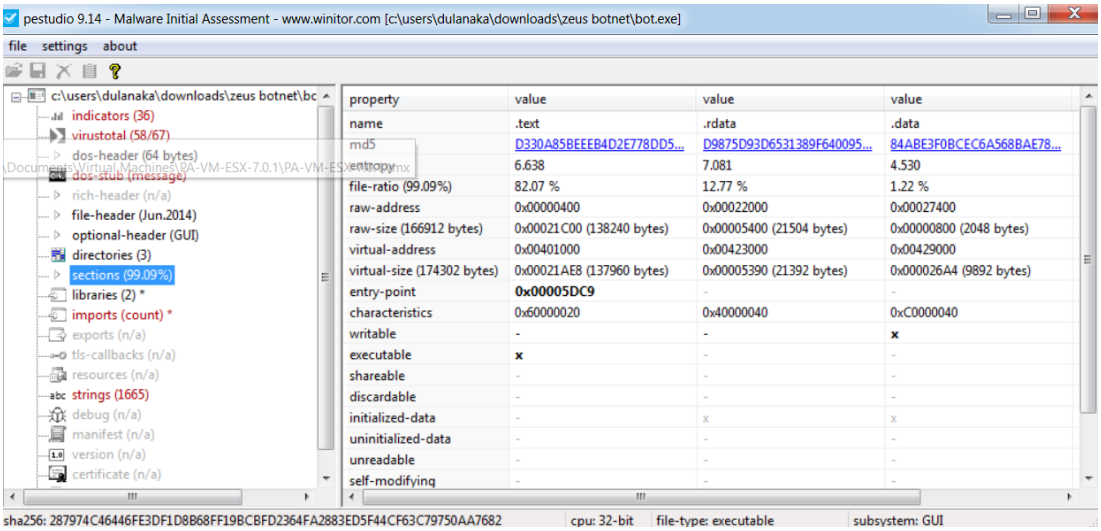


Fig. 13. Pestudio section tab

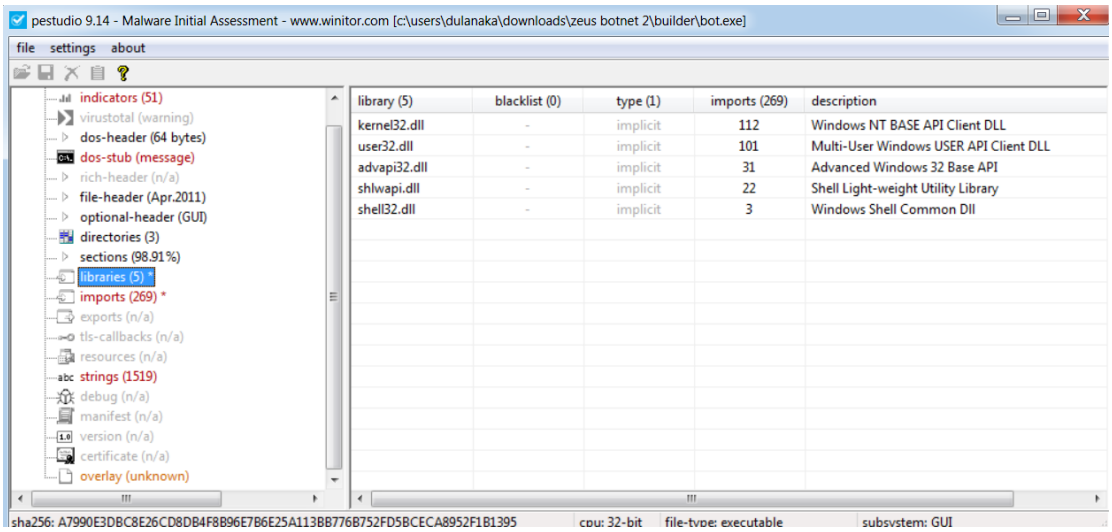


Fig. 14. Pestudio Libraries tab

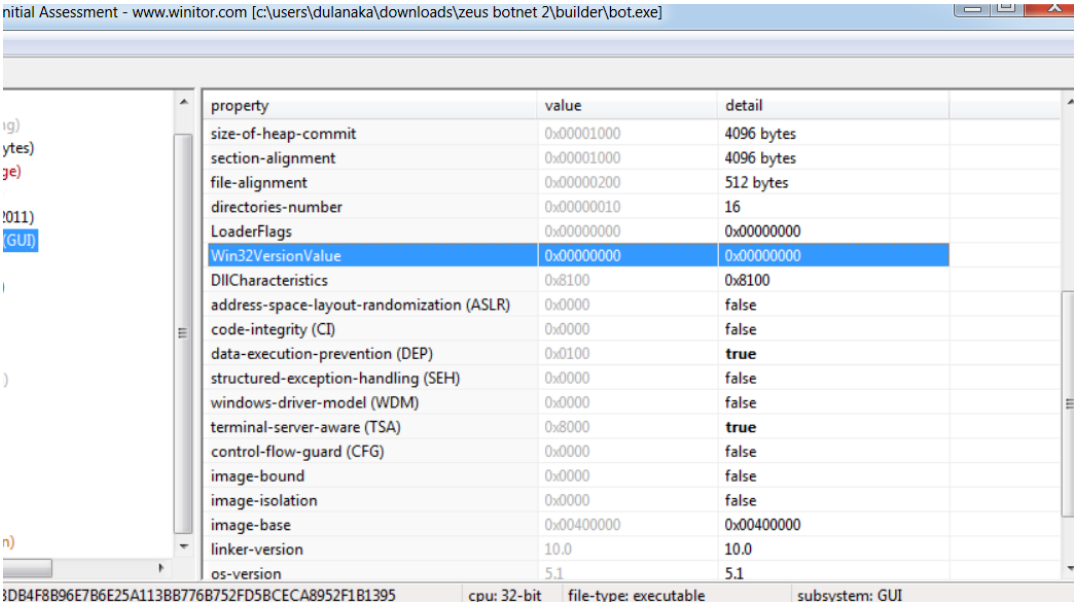


Fig. 15. Pestudio optional header GUI tab

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.

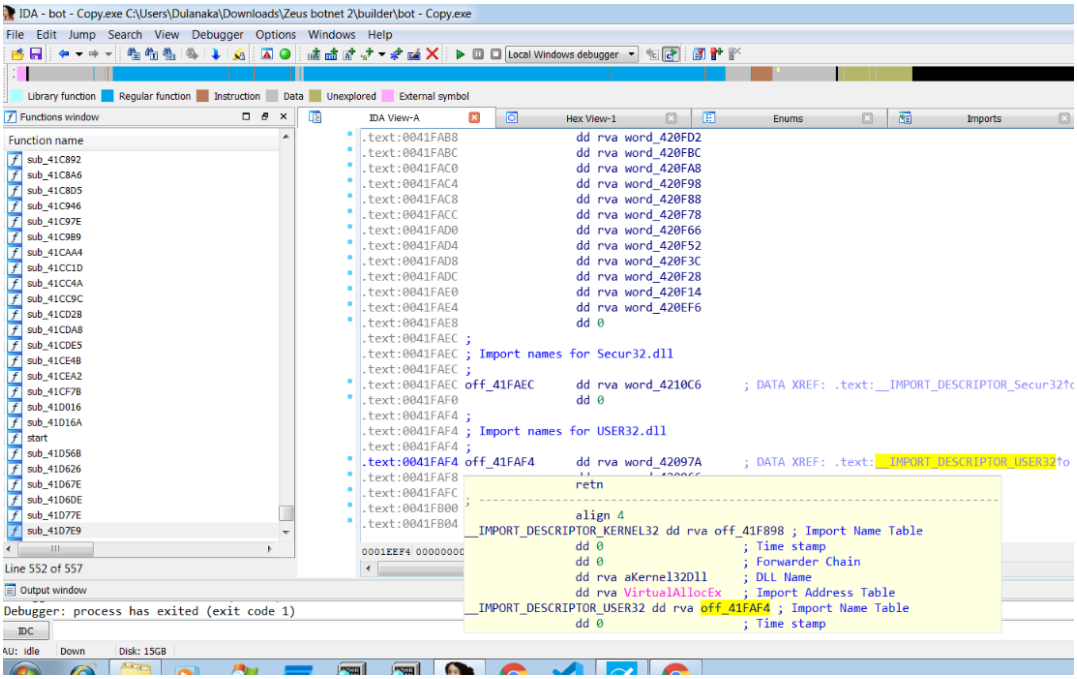


Fig. 16. Kernel32.dll and User32.dll files

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.

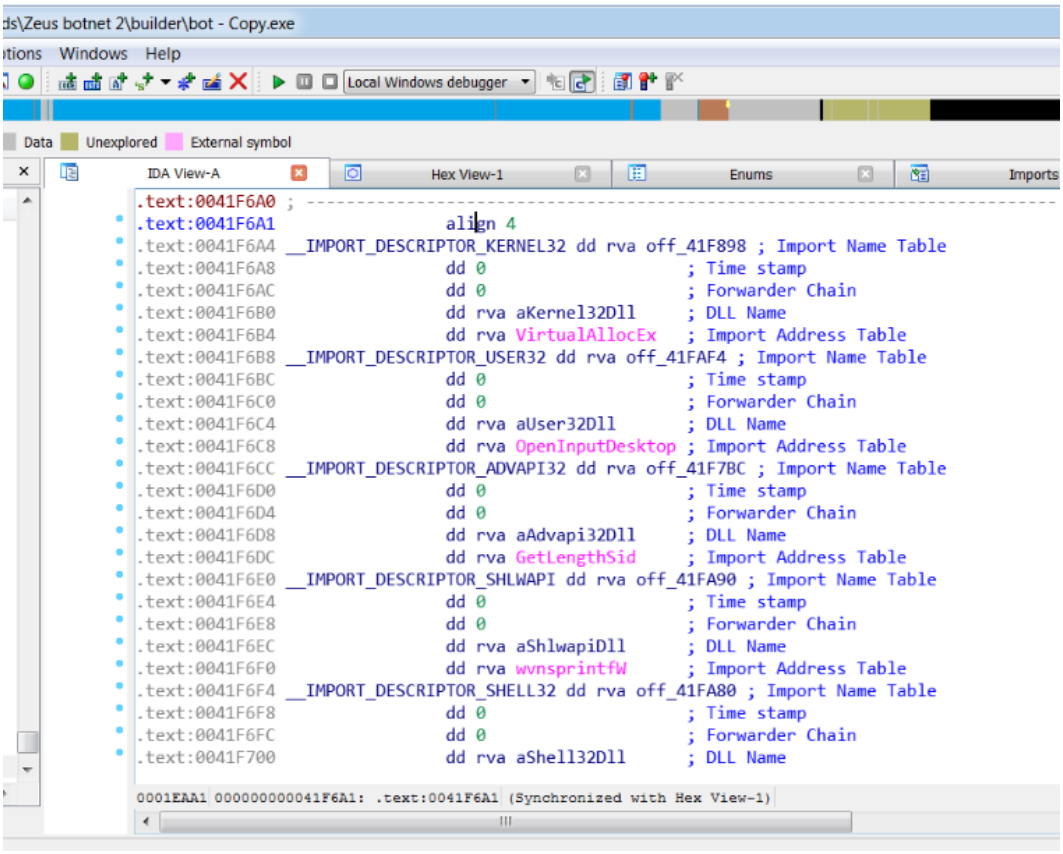


Fig. 17. loaded libraries

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.

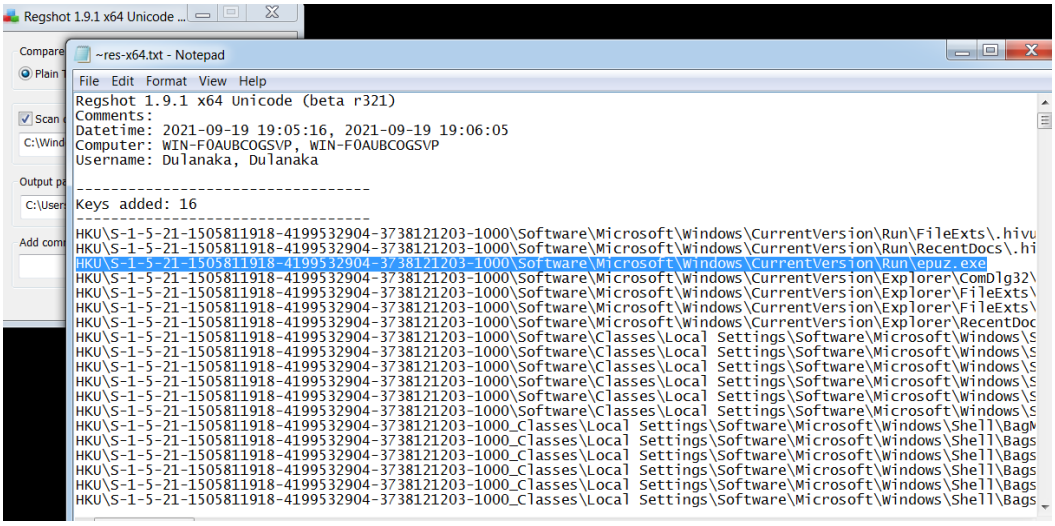


Fig. 18. Regshot output

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 'I' 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.

System Idle Process	0	94.06	0 K	28 K	
System	4		0 K	68 K	
Interrupts	n/a	2.97	0 K	0 K	Hardware Interrupts and DPCs
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 Applicat... Microsoft Corporation
services.exe	640		3,248 K	940 K	Services and Controller app Microsoft Corporation
vmacthlp.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
lsass.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.
procexp.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

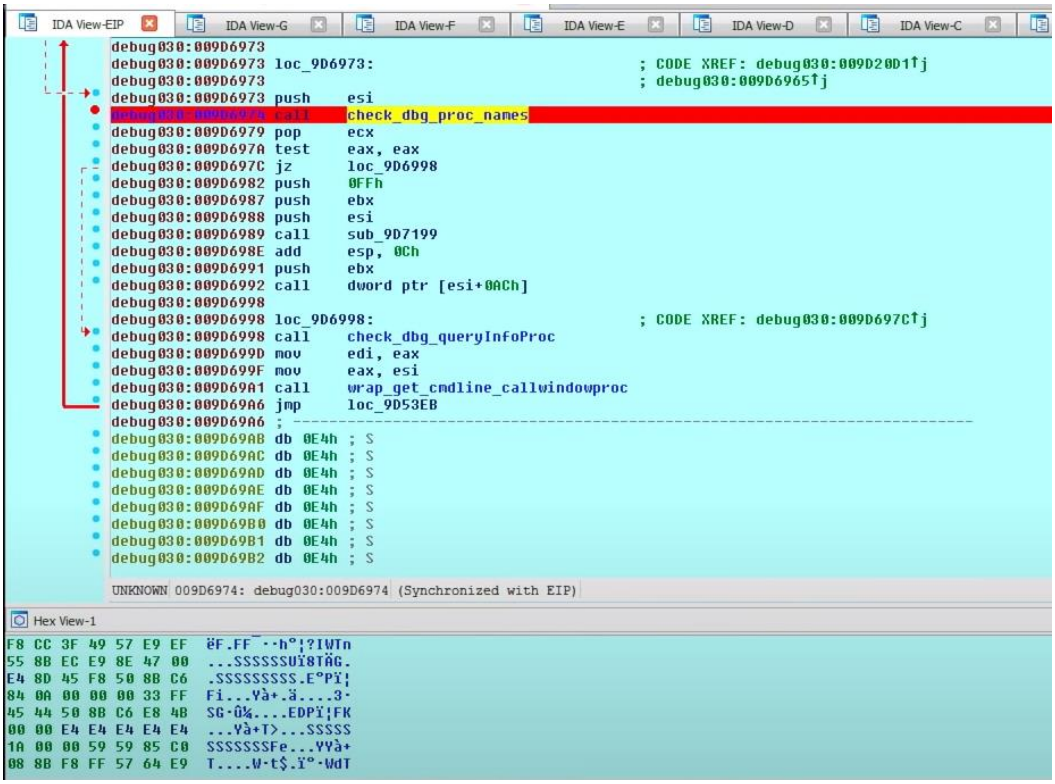


Fig.20 Debugging

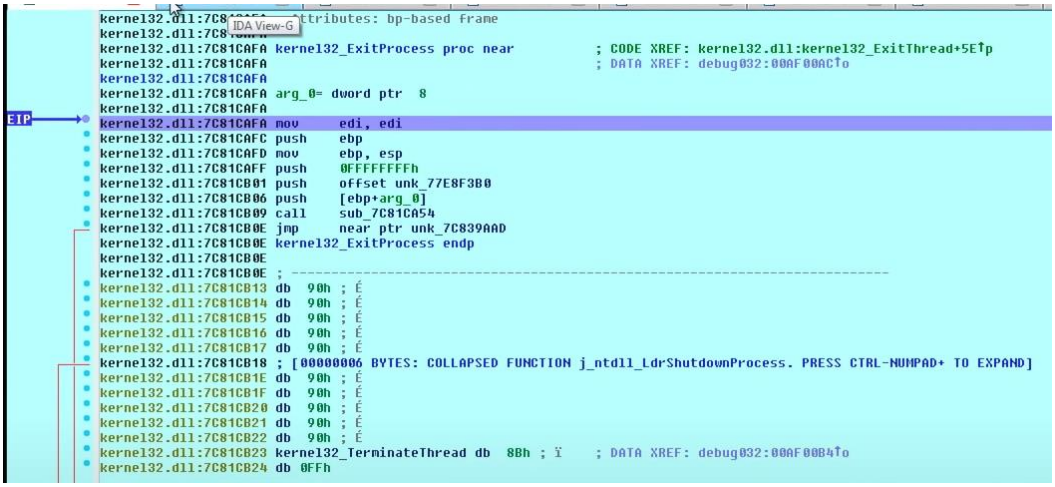


Fig.21 Debugging

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.

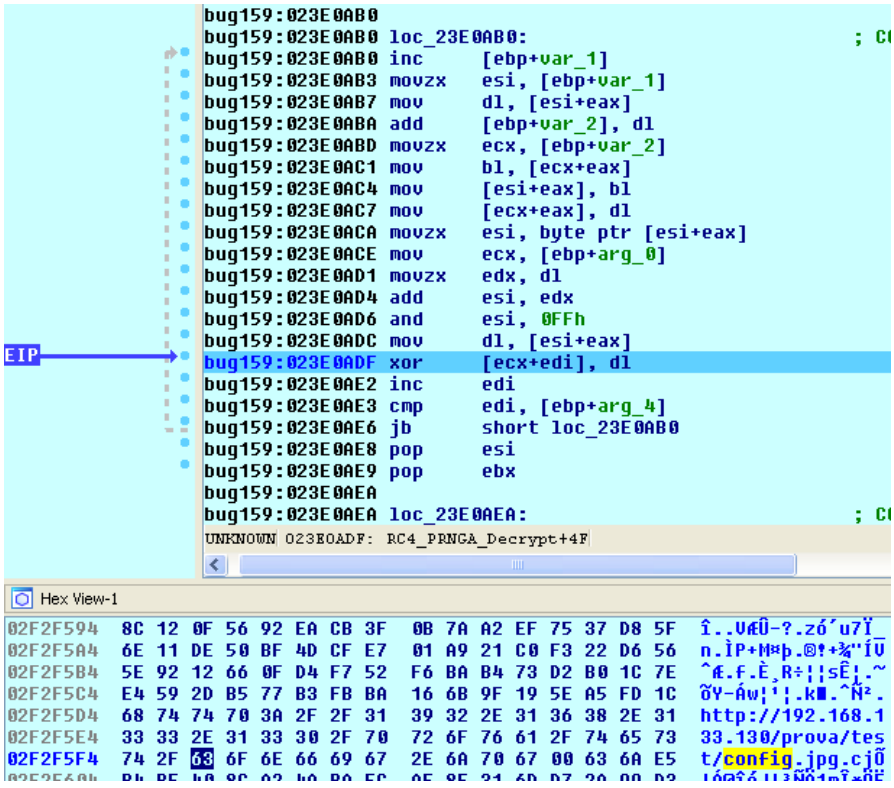


Fig.22. URL decryption

Continuing the study of the VM functions executed during the dynamiccofig 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.

config.jpg

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

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.

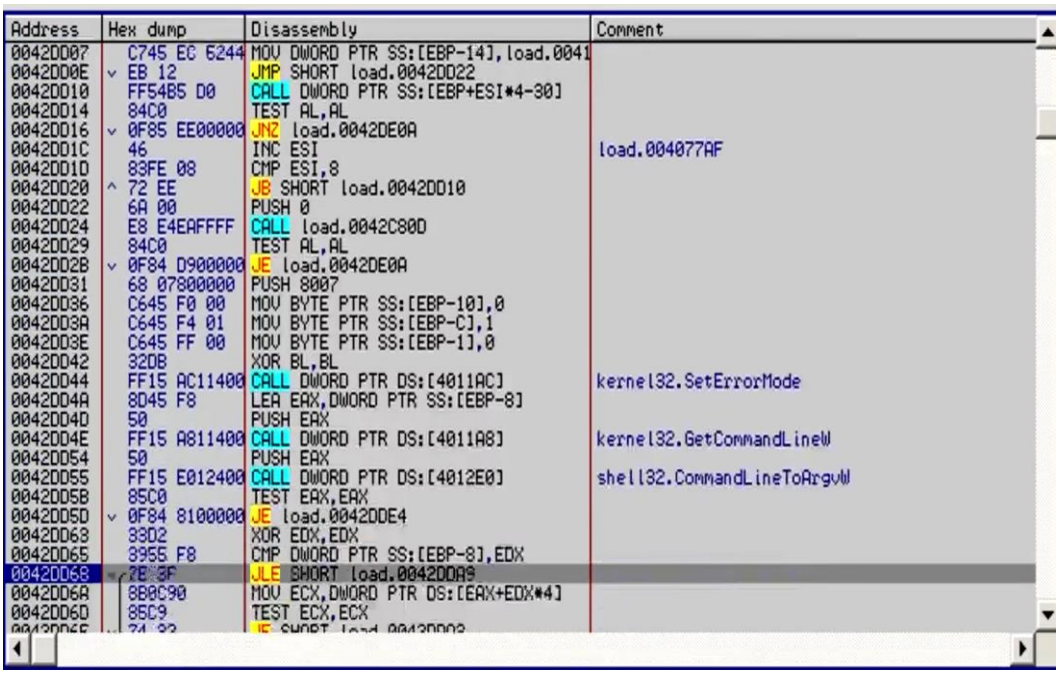


Fig. 24. Inside the Zeus layer analyze by Ollydbg

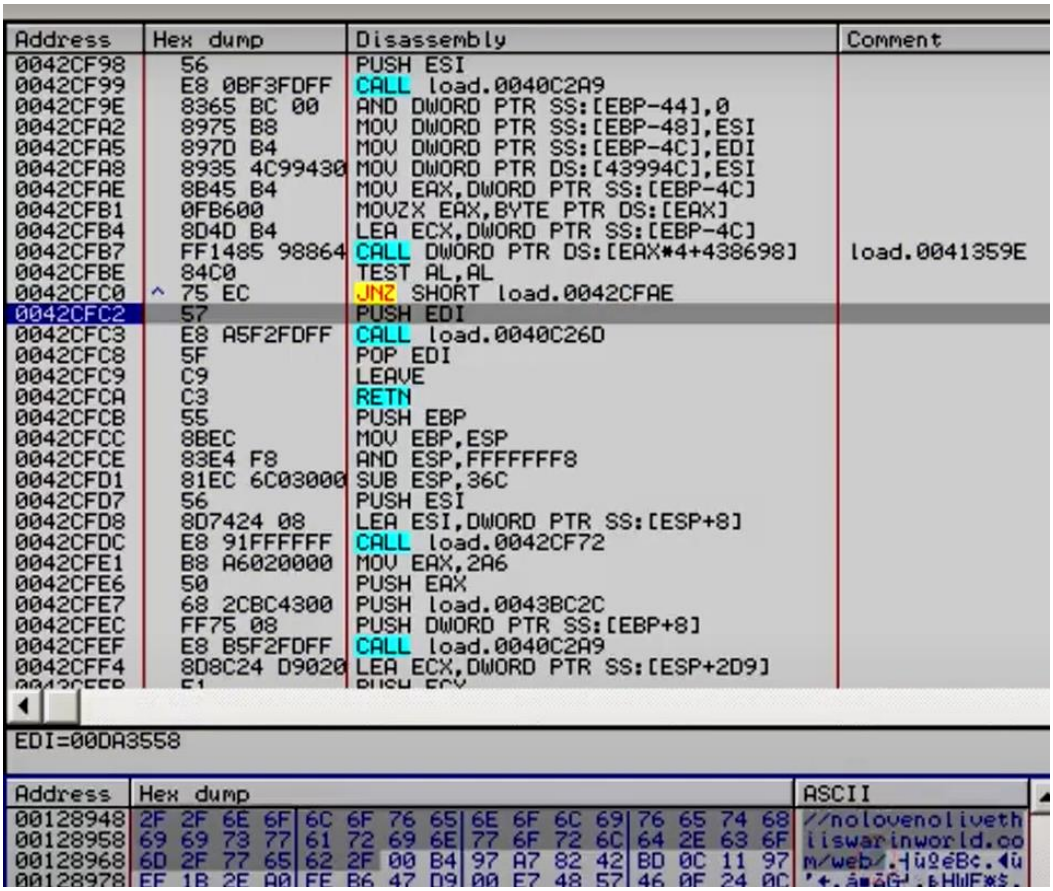


Fig.25. Get the version (Callback/user-agent/bot)

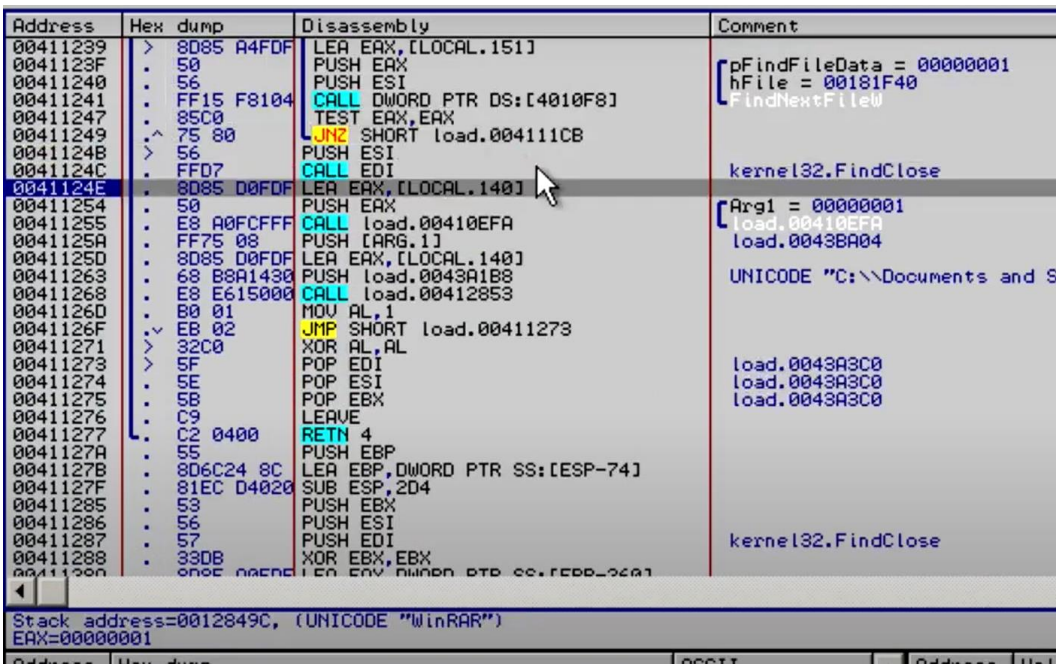


Fig.26. Found the folder location path for dropping

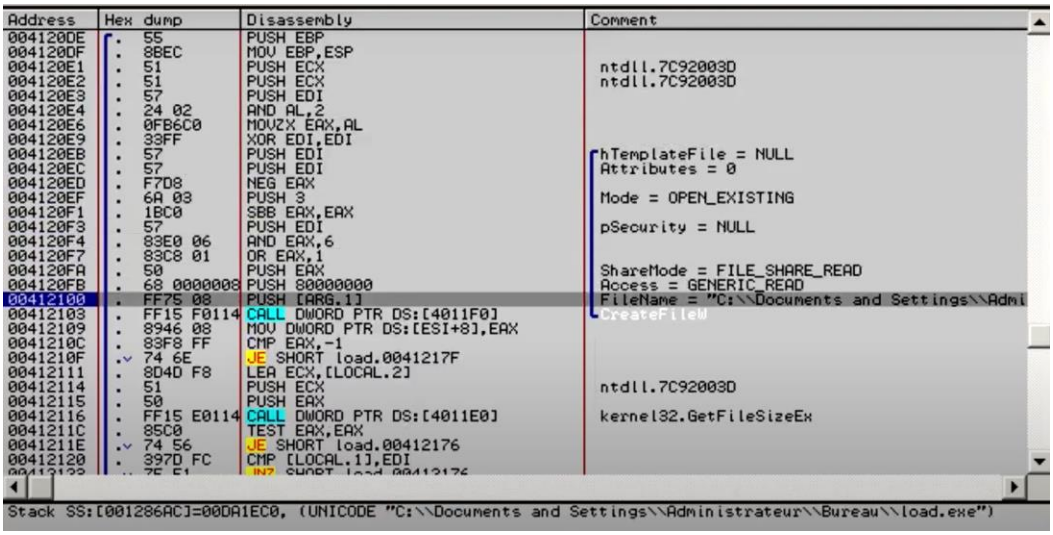


Fig.27. Calculate the bot location (load.exe)

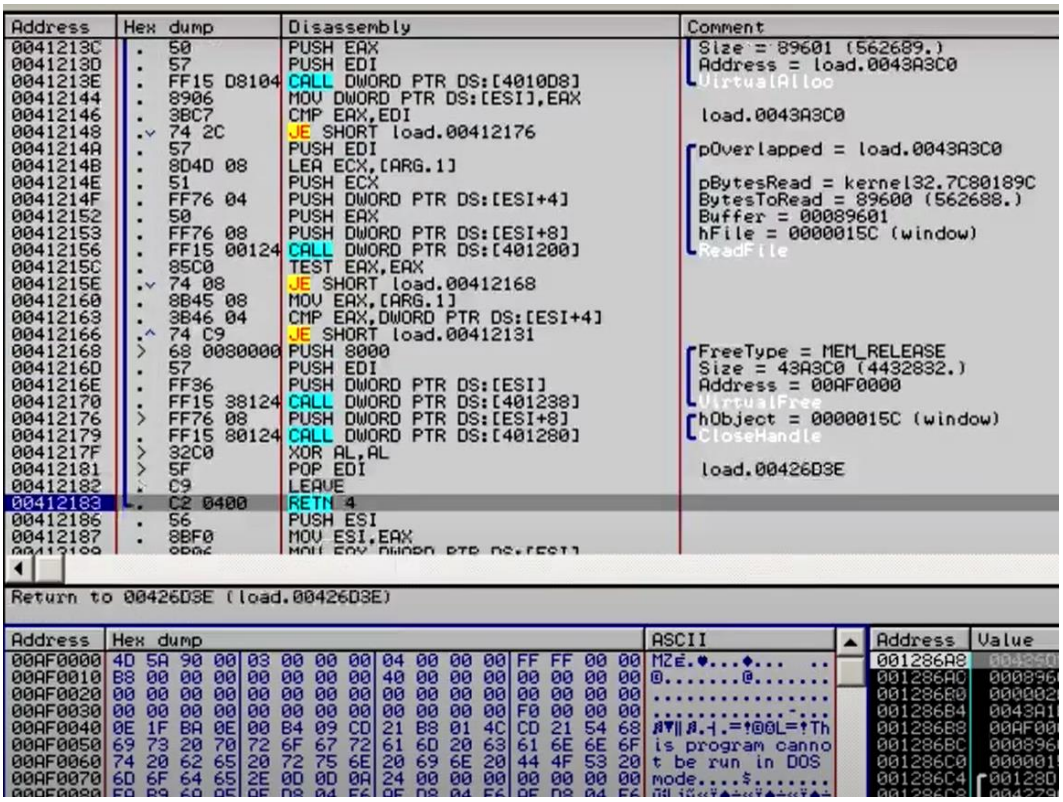


Fig.28. Checking the access for reading

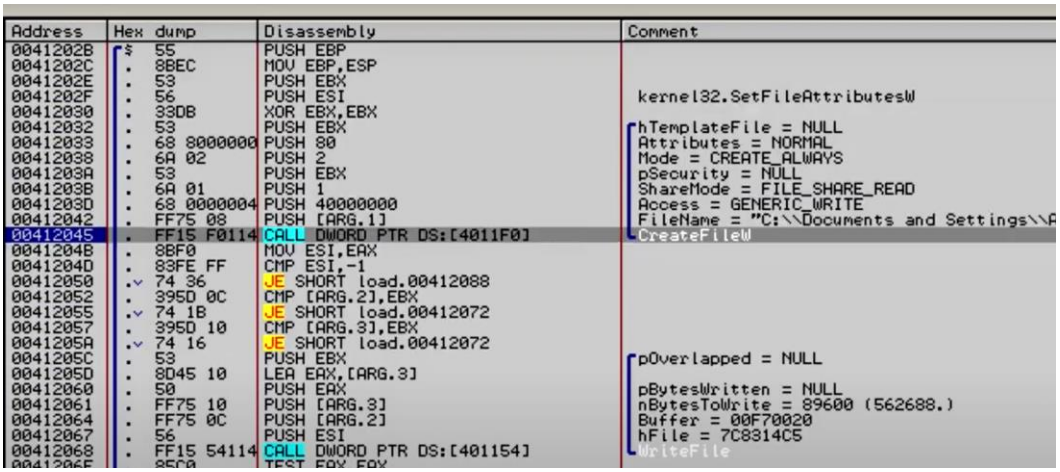


Fig.29. Write in %APPDATA% Winraragent.exe

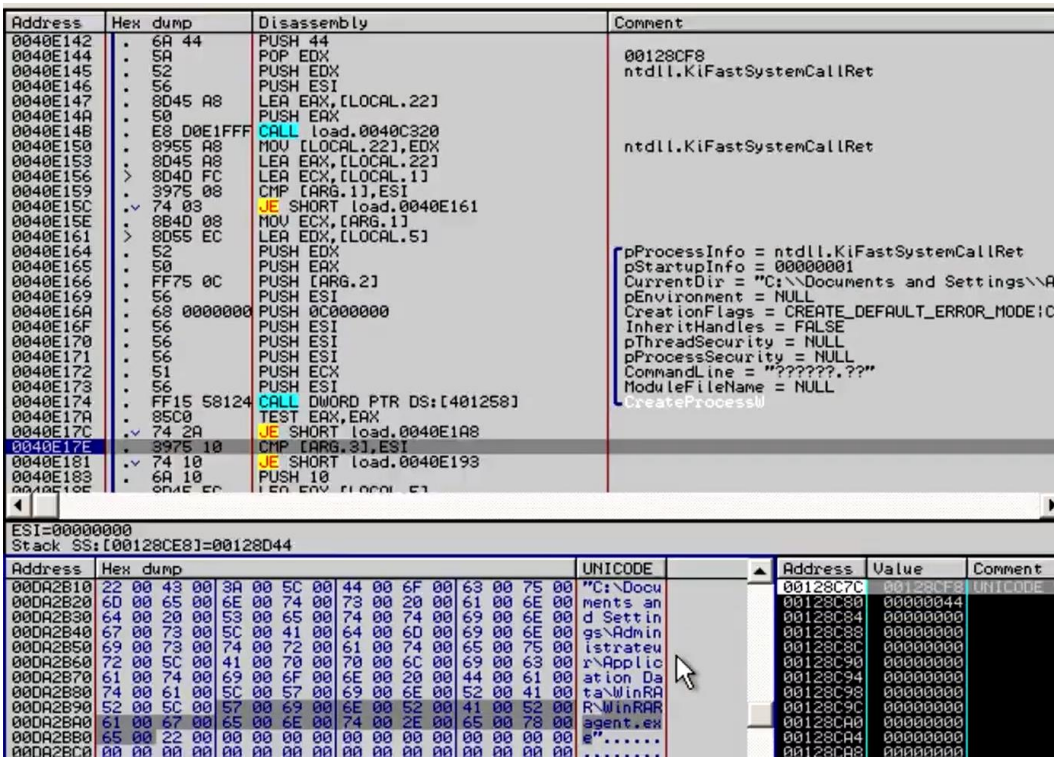


Fig.30. Execute the copy

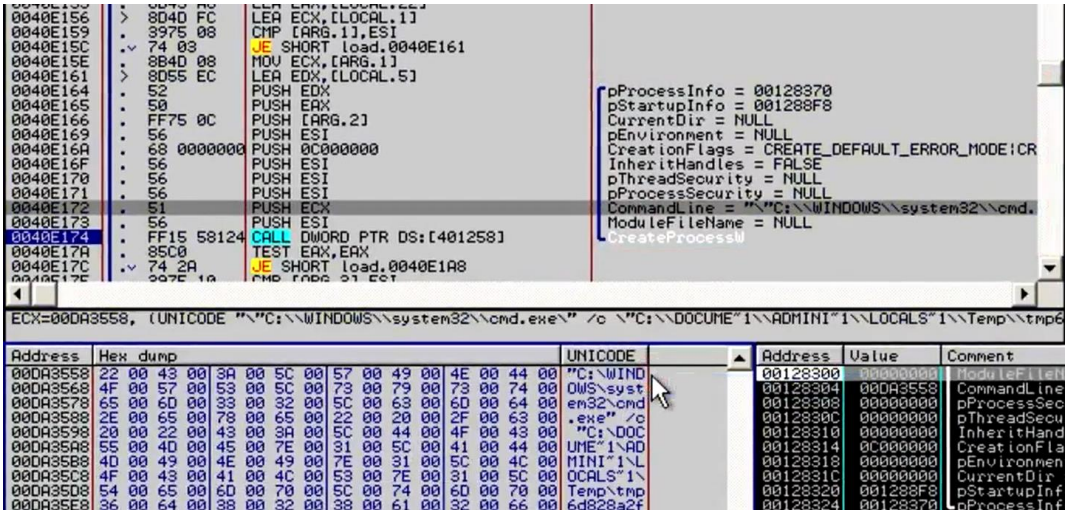


Fig.31. Execute the script using cmd.exe

After the terminated, going to do Injection

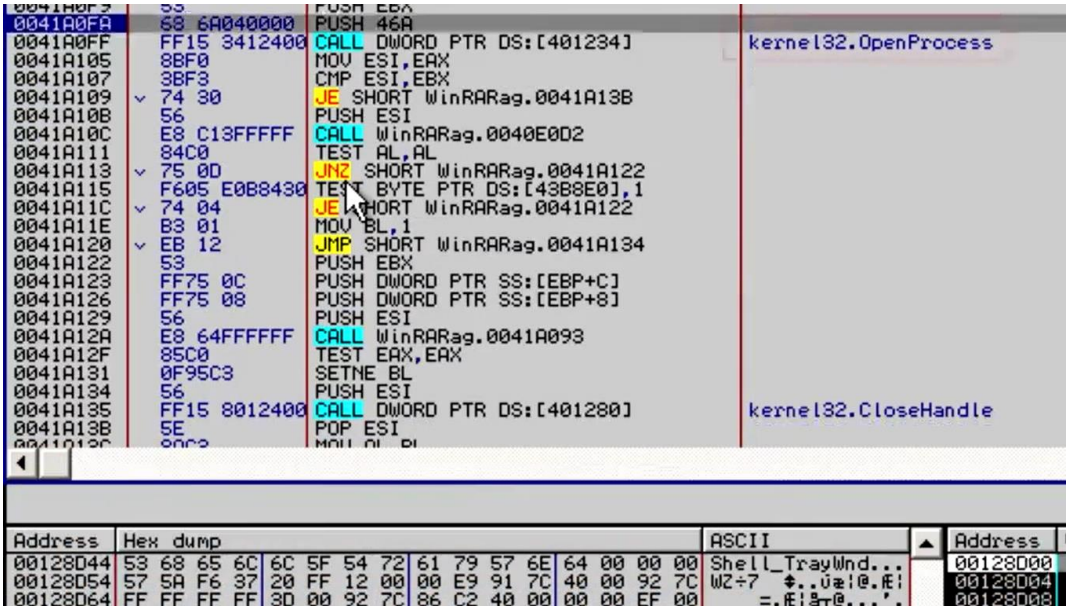


Fig.32. Find the process with class name 'Shell_Traywnd'

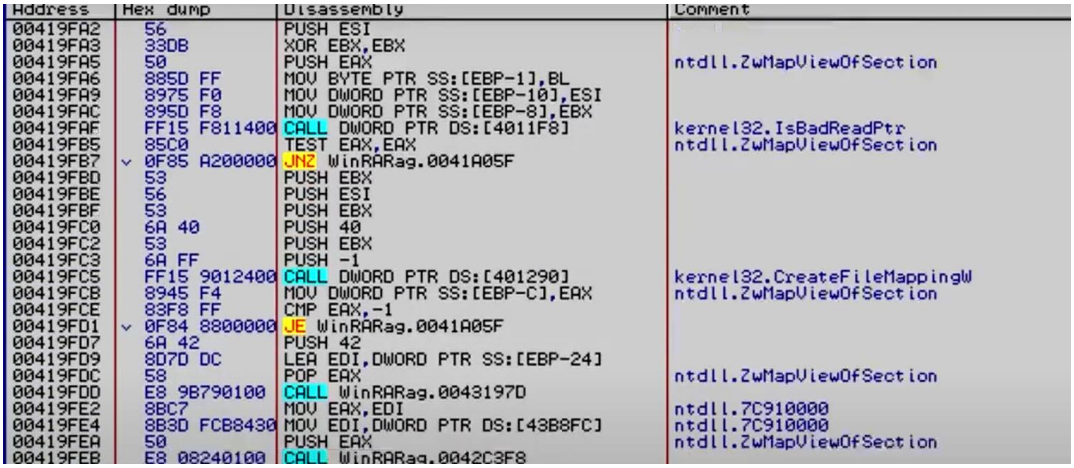


Fig.33. Inject using 'ZwMapViewOfSection'

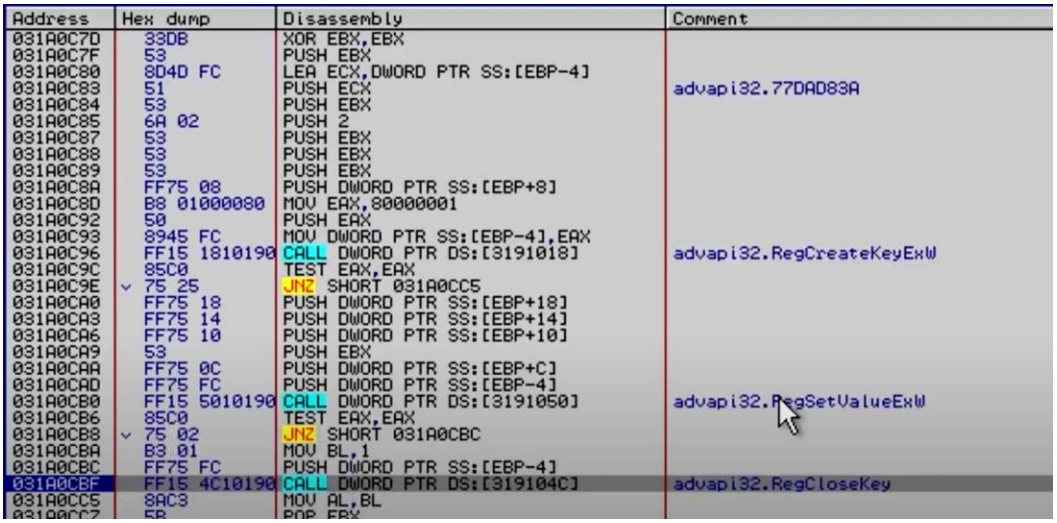


Fig.34. New thread starts

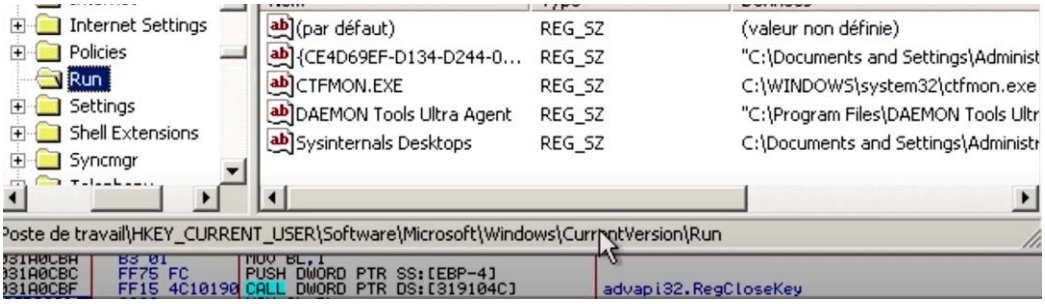
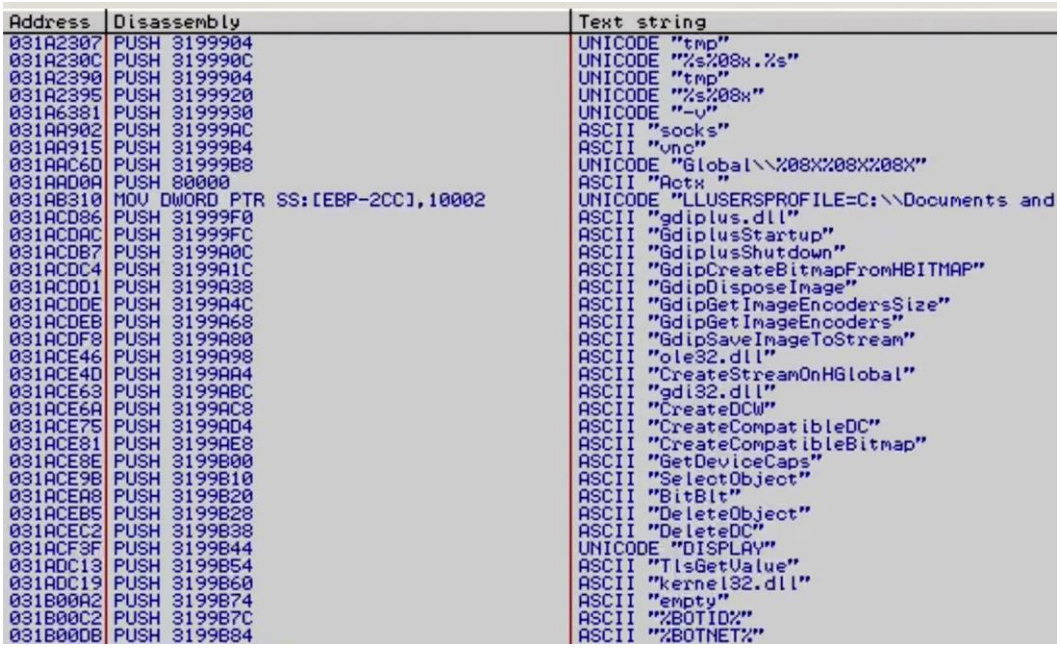


Fig.35. achieved via new thread starting in registry



Address	Disassembly	Text string
031A2307	PUSH 3199904	UNICODE "tmp"
031A230C	PUSH 319990C	UNICODE "%s%08x.%s"
031A2390	PUSH 3199904	UNICODE "tmp"
031A2395	PUSH 3199920	UNICODE "%s%08x"
031A6381	PUSH 3199930	UNICODE "-u"
031AA902	PUSH 31999AC	ASCII "socks"
031AA915	PUSH 31999B4	ASCII "vnc"
031AAC6D	PUSH 31999B8	UNICODE "Global\\%08X%08X%08X"
031AAD0A	PUSH 80000	ASCII "Actx "
031AB310	MOV DWORD PTR SS:[EBP-2CC],10002	UNICODE "LLUSERSPROFILE=C:\\Documents and
031ACD86	PUSH 31999F0	ASCII "gdiplus.dll"
031ACDAC	PUSH 31999FC	ASCII "GdiplusStartup"
031ACDB7	PUSH 3199A0C	ASCII "GdiplusShutdown"
031ACDC4	PUSH 3199A1C	ASCII "GdipCreateBitmapFromHBITMAP"
031ACDD1	PUSH 3199A38	ASCII "GdipDisposeImage"
031ACDDE	PUSH 3199A4C	ASCII "GdipGetImageEncodersSize"
031ACDEB	PUSH 3199A68	ASCII "GdipGetImageEncoders"
031ACDF8	PUSH 3199A80	ASCII "GdipSaveImageToStream"
031ACE46	PUSH 3199A98	ASCII "ole32.dll"
031ACE4D	PUSH 3199AA4	ASCII "CreateStreamOnHGlobal"
031ACE63	PUSH 3199ABC	ASCII "gdi32.dll"
031ACE6A	PUSH 3199AC8	ASCII "CreateDCW"
031ACE75	PUSH 3199AD4	ASCII "CreateCompatibleDC"
031ACE81	PUSH 3199AE8	ASCII "CreateCompatibleBitmap"
031ACE8E	PUSH 3199B00	ASCII "GetDeviceCaps"
031ACE9B	PUSH 3199B10	ASCII "SelectObject"
031ACEA8	PUSH 3199B20	ASCII "BitBlt"
031ACEB5	PUSH 3199B28	ASCII "DeleteObject"
031ACEC2	PUSH 3199B38	ASCII "DeleteDC"
031ACF3F	PUSH 3199B44	UNICODE "DISPLAY"
031ADC13	PUSH 3199B54	ASCII "TlsGetValue"
031ADC19	PUSH 3199B60	ASCII "kernel32.dll"
031B00A2	PUSH 3199B74	ASCII "empty"
031B00C2	PUSH 3199B7C	ASCII "%BOTID%"
031B00DB	PUSH 3199B84	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 webinejct'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.

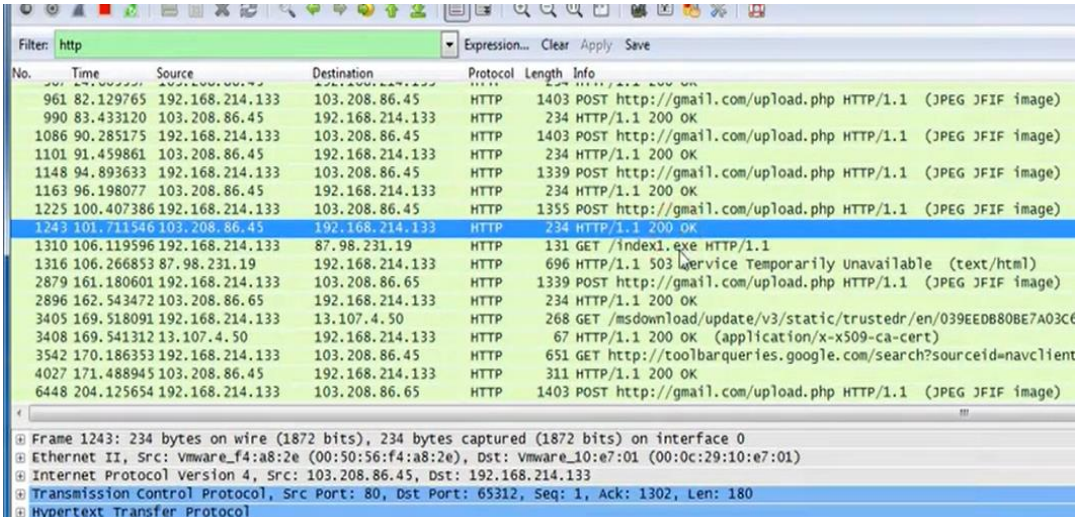


Fig. 37. Wireshark analysis

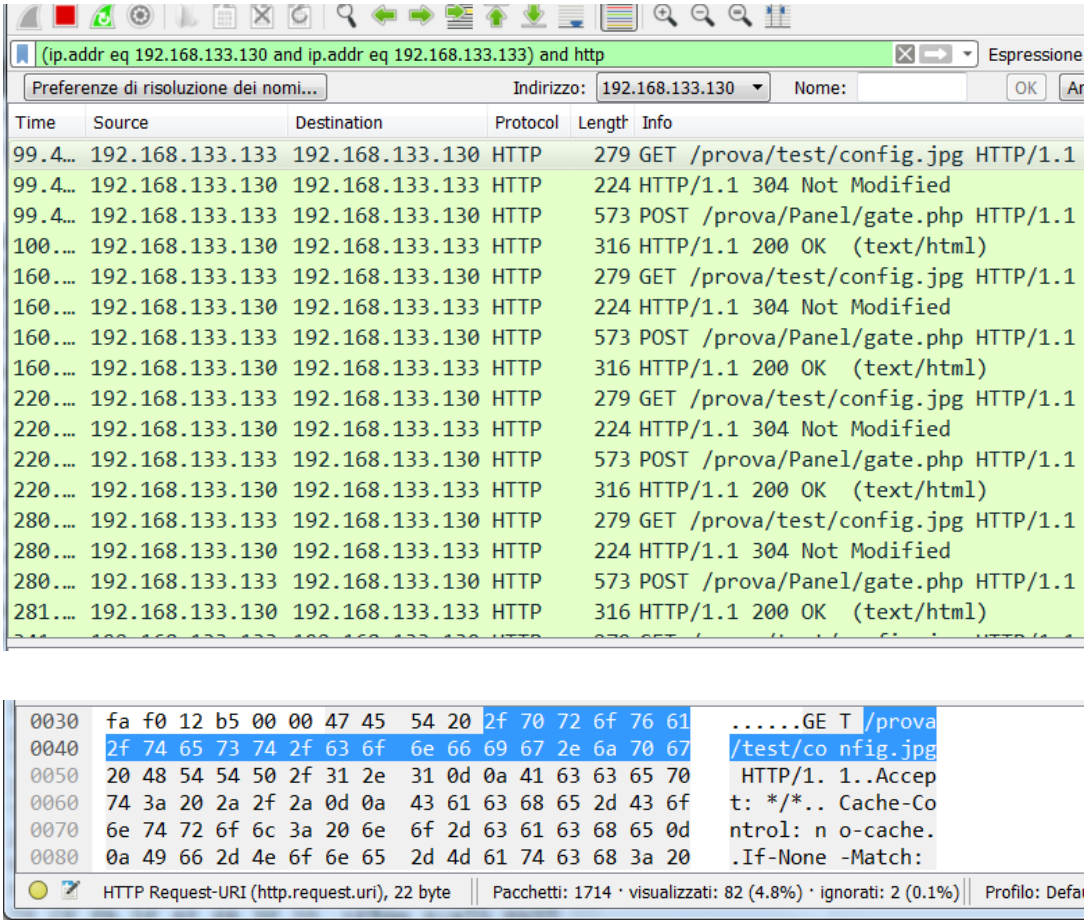


Fig. 38. Wireshark analysis

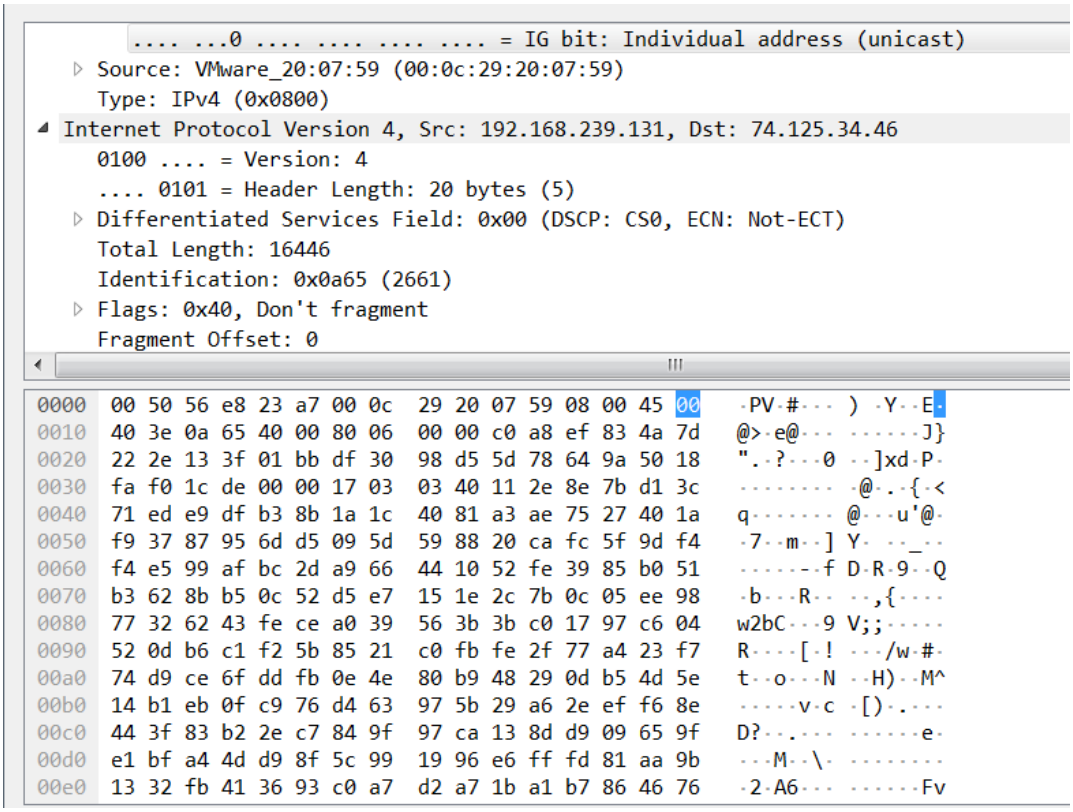


Fig. 39. Wireshark analysis

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).

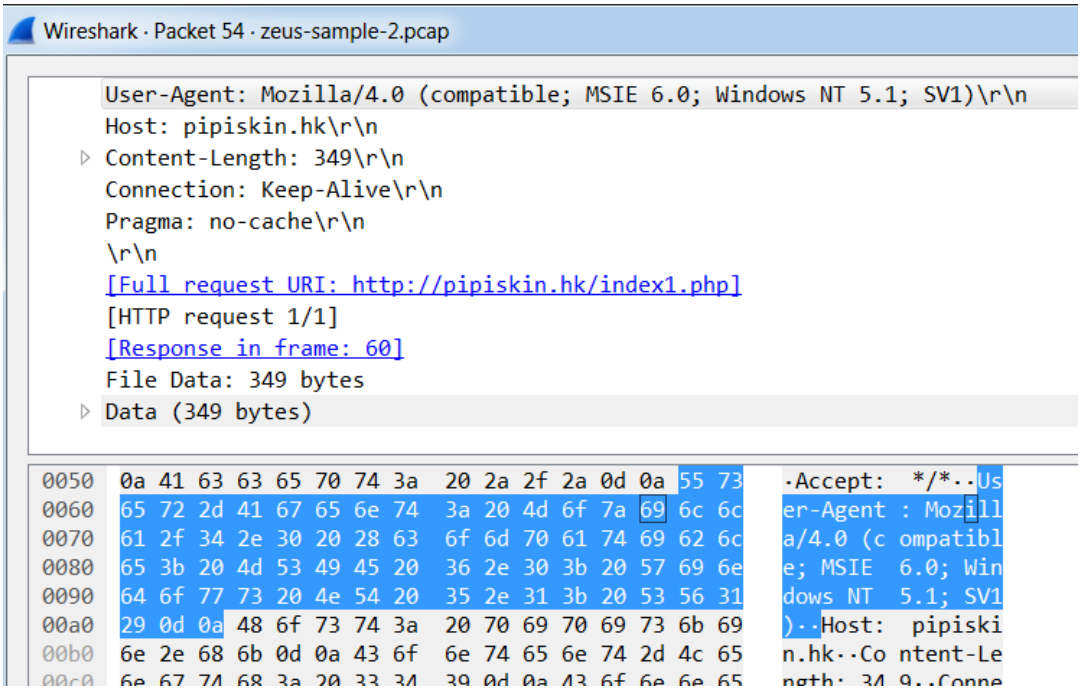


Fig. 40. Wireshark analysis

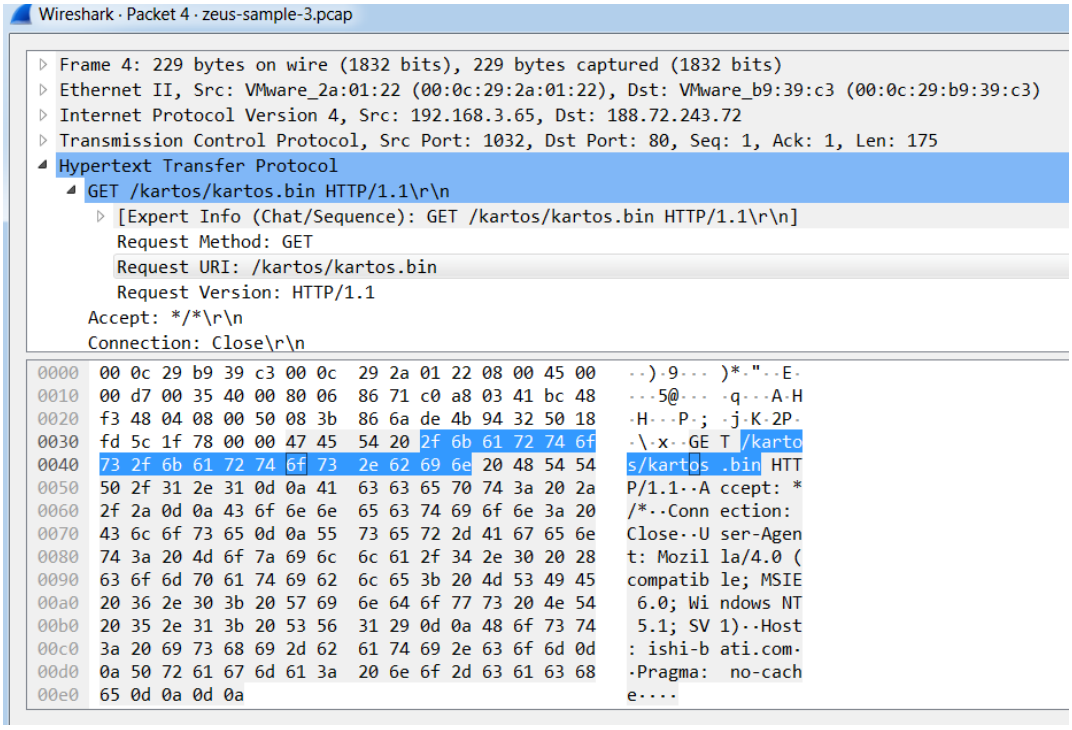


Fig. 41. Wireshark analysis

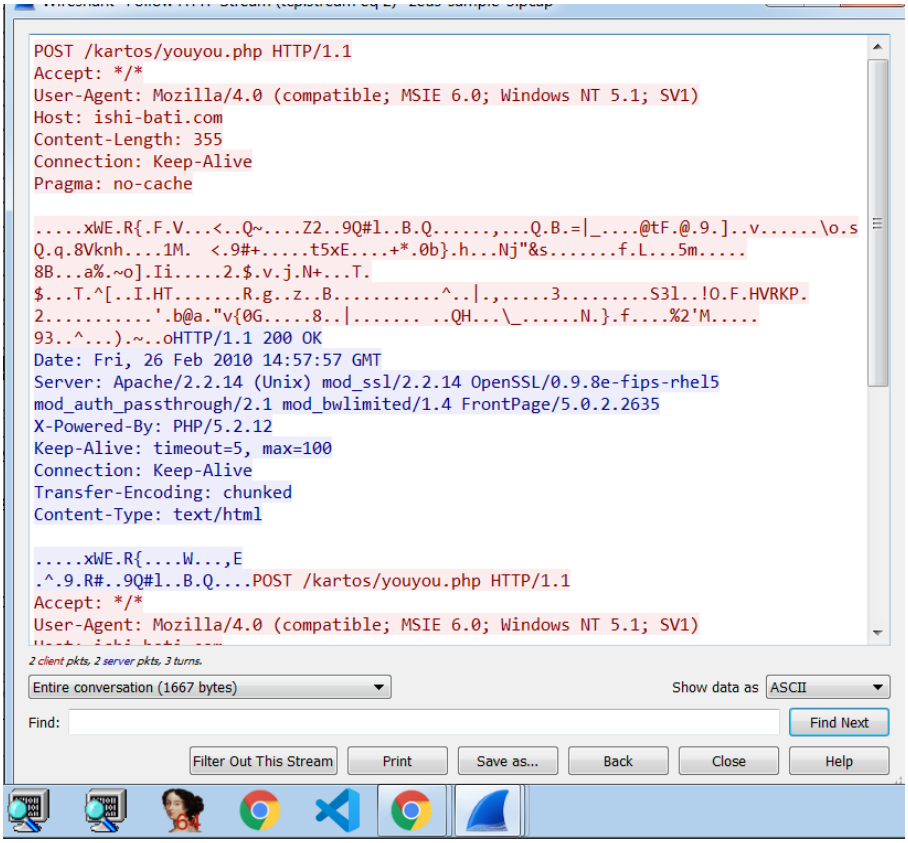


Fig. 42. Wireshark analysis

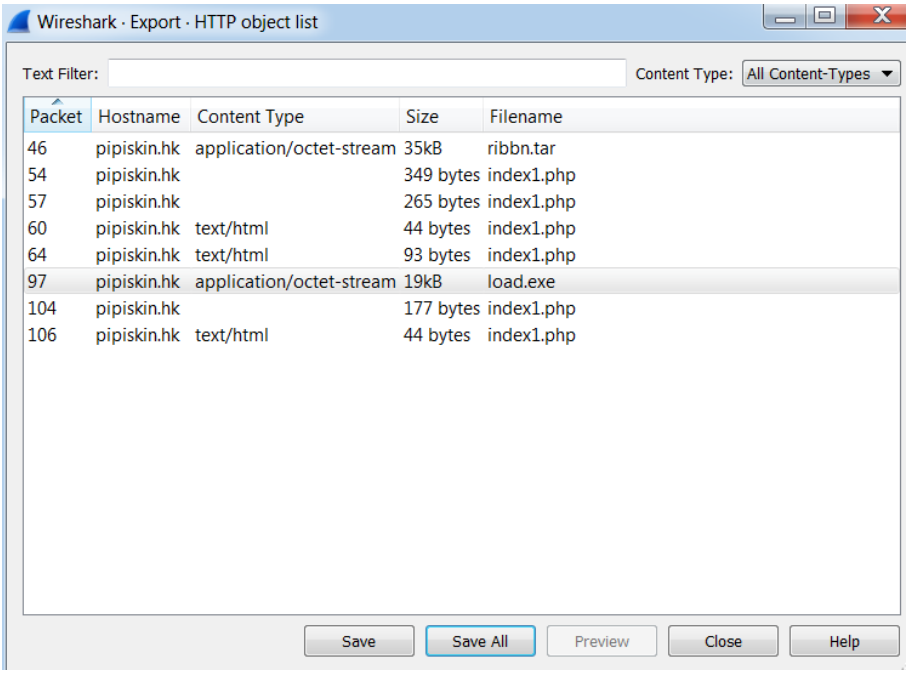


Fig. 43. Wireshark analysis

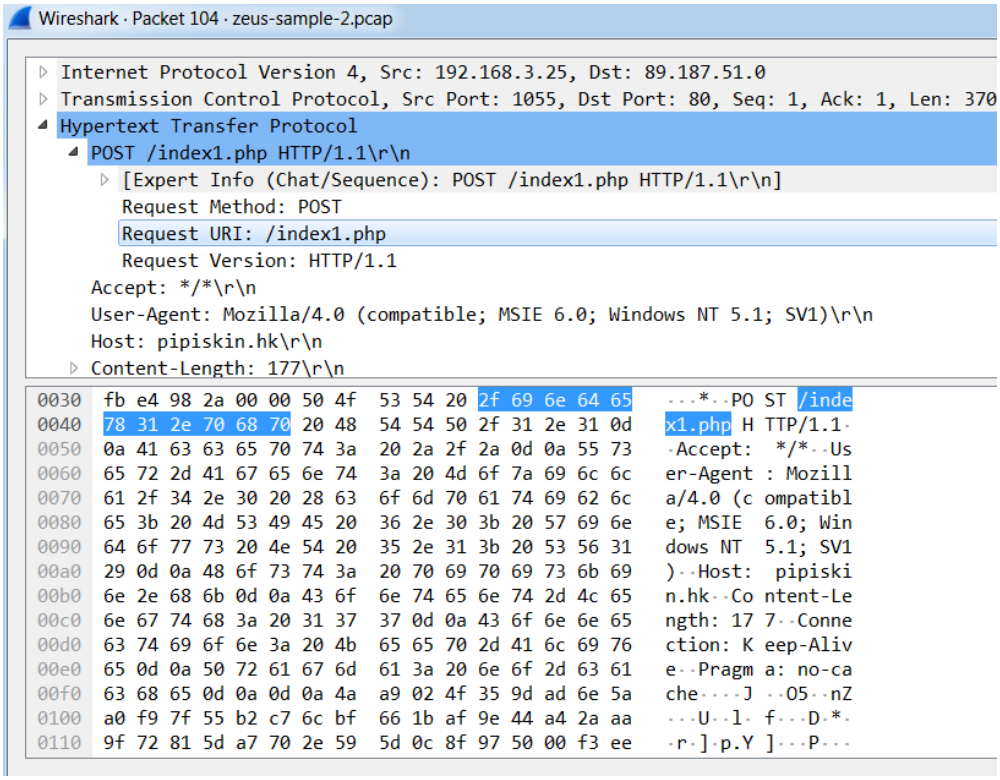


Fig. 44. Wireshark analysis

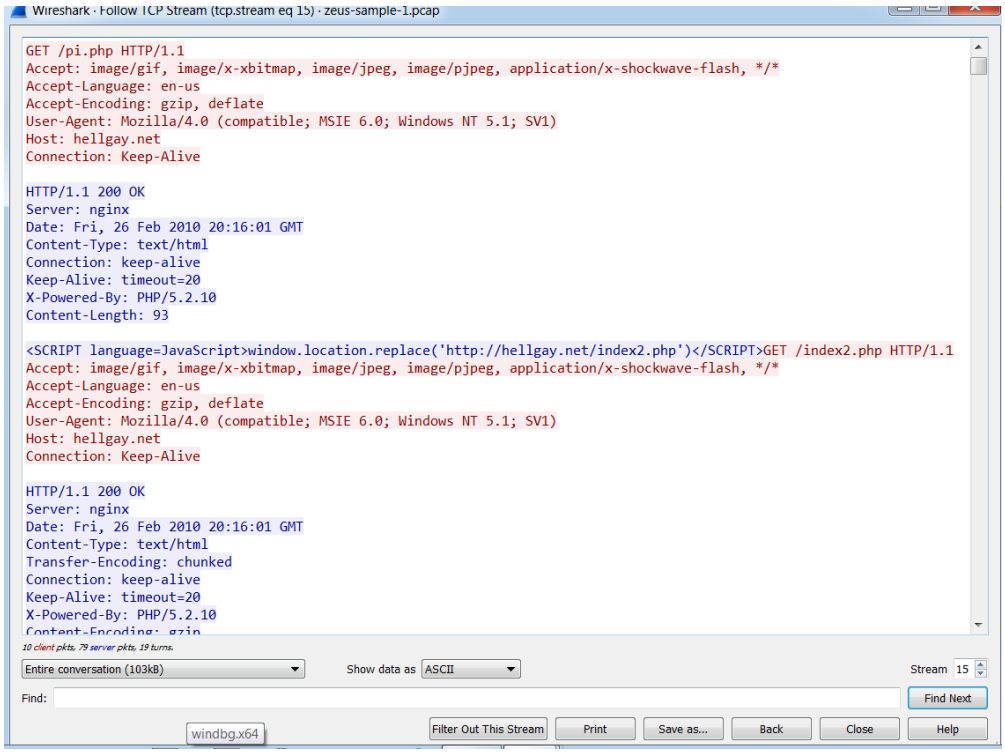


Fig. 45. Wireshark analysis

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.

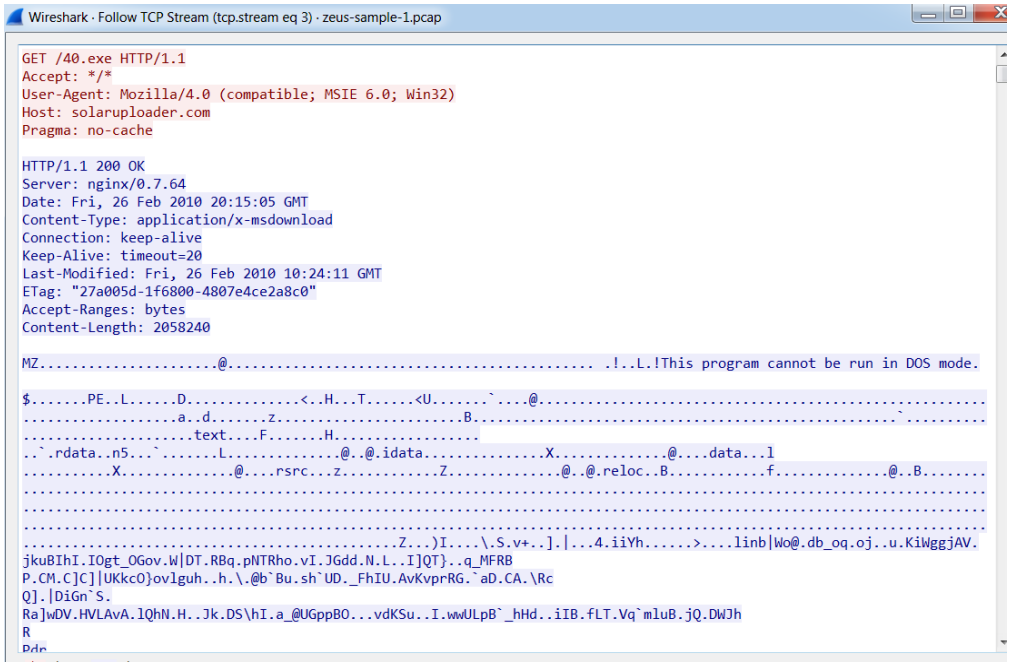


Fig. 46. Wireshark analysis

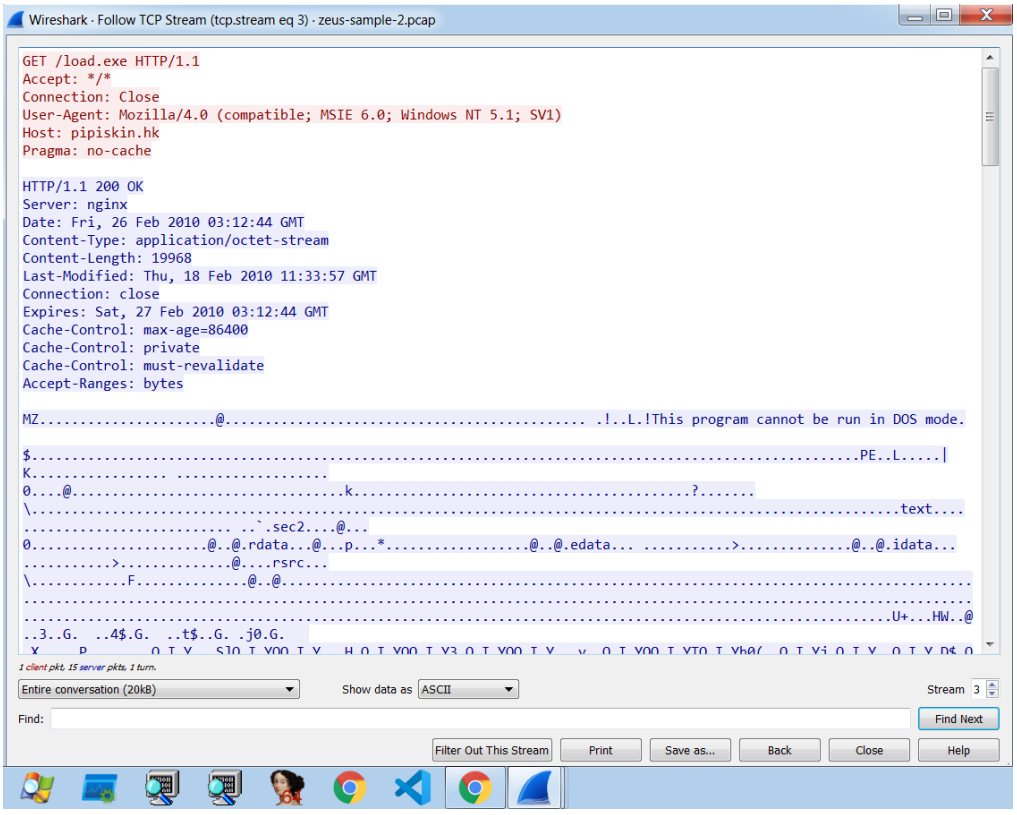


Fig. 47. Wireshark analysis

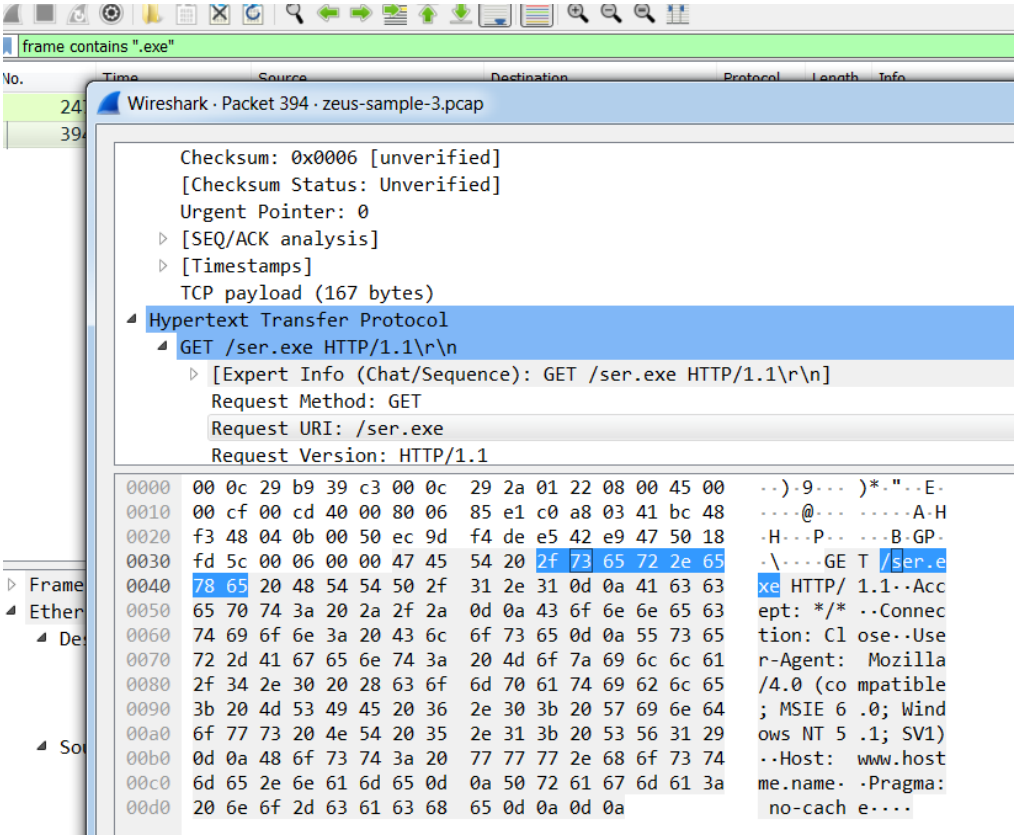


Fig. 48. Wireshark analysis

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).

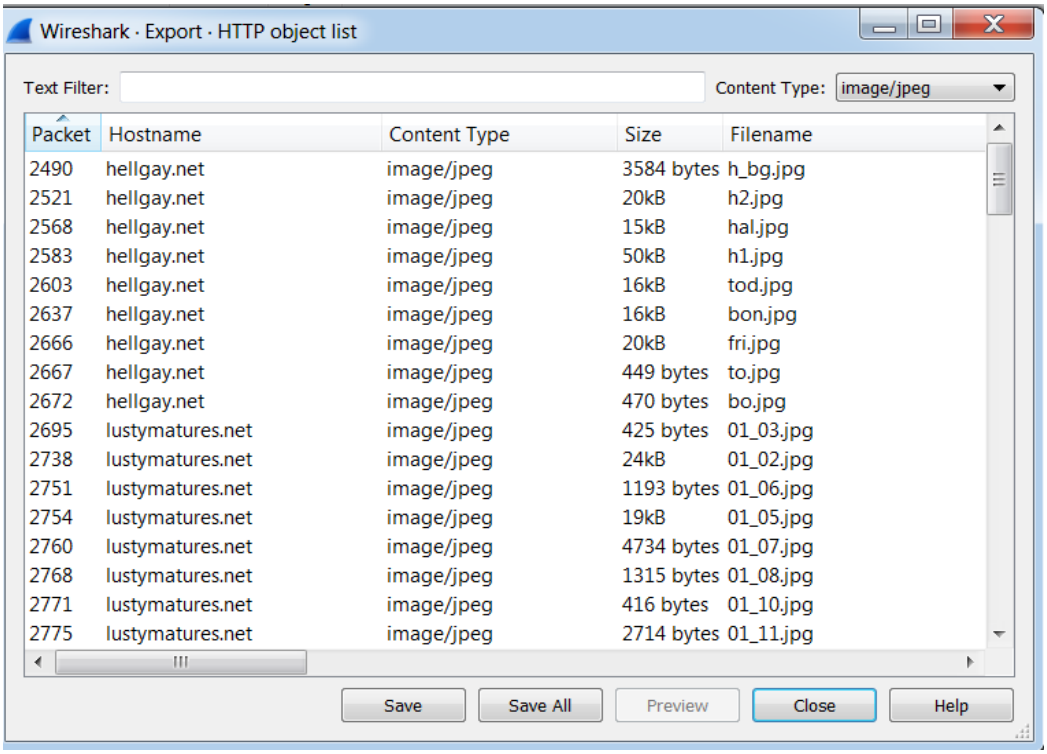


Fig. 49. Wireshark analysis

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.

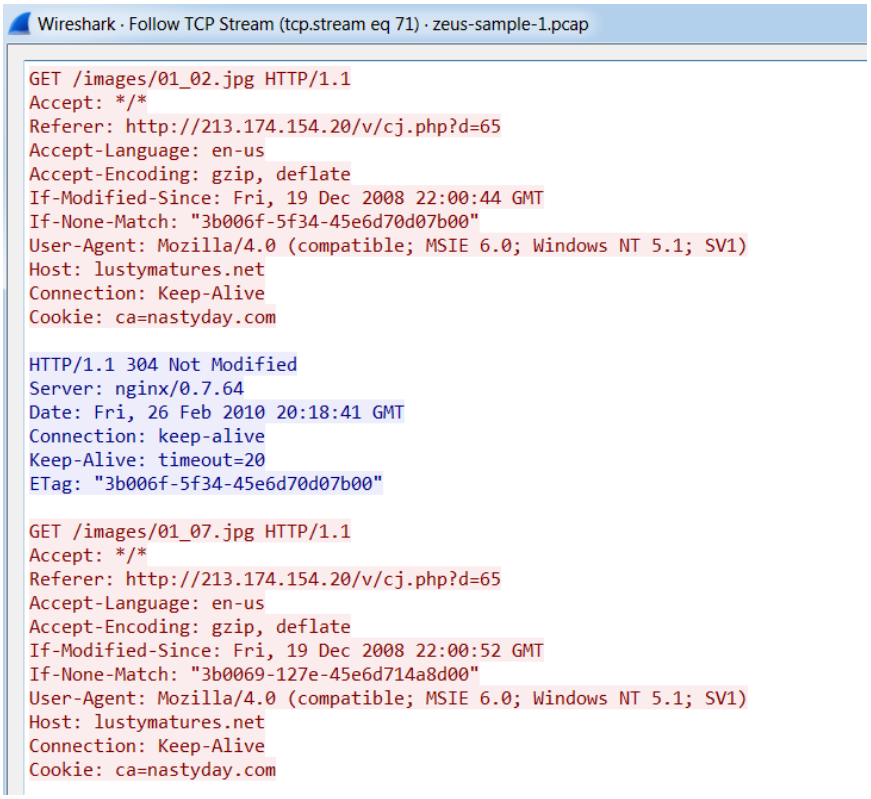


Fig. 50. Wireshark analysis

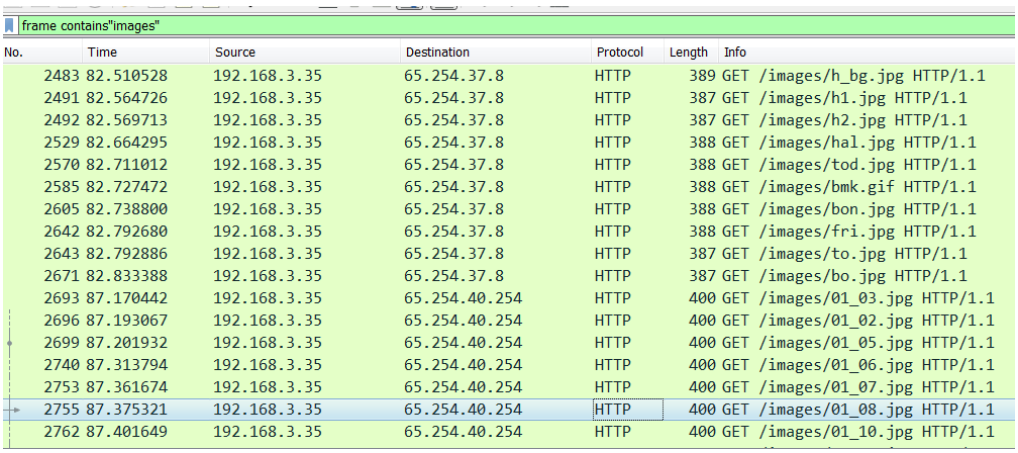


Fig. 51. Wireshark analysis

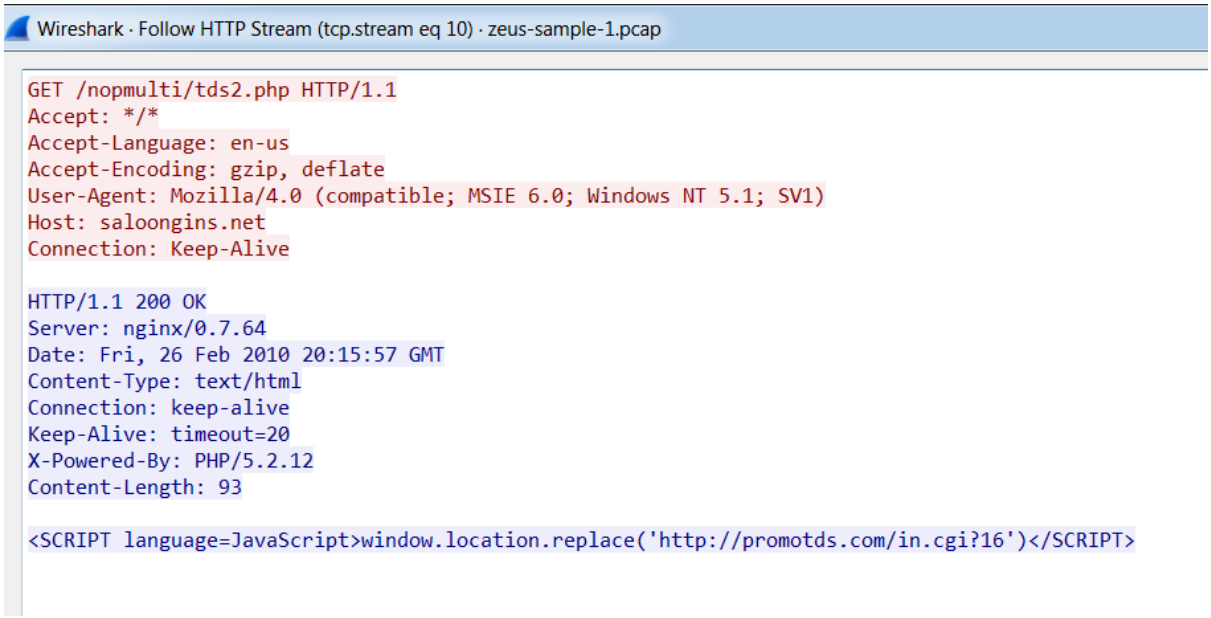


Fig. 52. Wireshark analysis

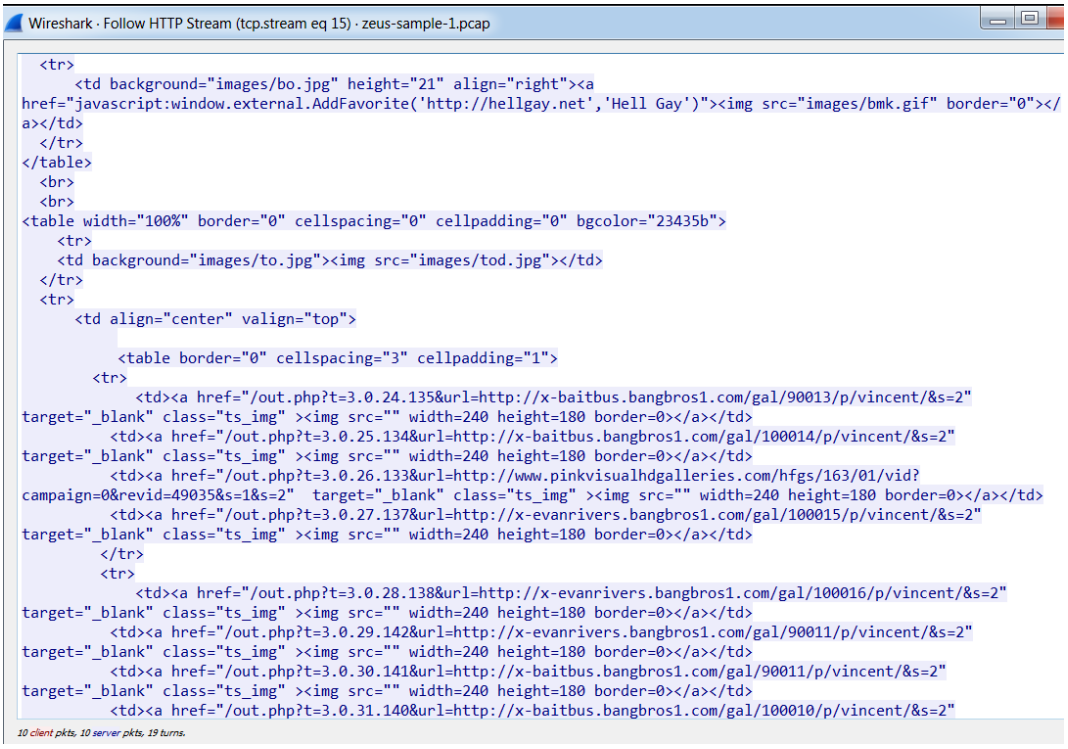


Fig. 53. HTTP Stream Wireshark analysis

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. REFFERENCES

[1] “VirsuTotal” [Online]. Available: https://www.virustotal.com/.

[2] “OllyDbg” [Online]. Available: http://www.ollydbg.de/.

[3] Josh Fruhlinger , “Malware explained: How to prevent, detect and recover from it” Accessed: May 17, 2019. [online]. Available: https://www.csoonline.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/.