On the Analysis of the Zeus Botnet Crimeware Toolkit

H. Binsalleeh †‡, T. Ormerod †‡, A. Boukhtouta †‡, P. Sinha †‡, A. Youssef †‡, M. Debbabi †‡, and L. Wang †‡
† National Cyber Forensics and Training Alliance Canada
‡ Computer Security Laboratory, Concordia University
Montreal, Quebec, Canada
{h_binsal,t_ormero,a_boukh,p_sinh,youssef,debbabi,wang}@ciise.concordia.ca

***Abstract*—In this paper, we present our reverse engineering results for the Zeus crimeware toolkit which is one of the recent and powerful crimeware tools that emerged in the Internet underground community to control botnets. Zeus has reportedly infected over 3.6 million computers in the United States. Our analysis aims at uncovering the various obfuscation levels and shedding the light on the resulting code. Accordingly, we explain the bot building and installation/infection processes. In addition, we detail a method to extract the encryption key from the malware binary and use that to decrypt the network communications and the botnet configuration information. The reverse engineering insights, together with network traffic analysis, allow for a better understanding of the technologies and behaviors of such modern HTTP botnet crimeware toolkits and opens an opportunity to inject falsified information into the botnet communications which can be used to defame this crimeware toolkit.**

I. Introduction

The tremendous growth in the use of Internet technologies in different walks of life has molded the living habits of most people. The traditional ways of trading and marketing, education, communication, and broadcasting are replaced by the innovative web-based applications and online systems. However, these Internet applications are abused by perpetrators and hackers for committing different kinds of crimes including spamming, phishing, and distributed denial of service (DDoS) attacks. In the majority of Internet mediated cyber crimes, the victimization tactics that are used vary from the simple anonymity to the identity theft and impersonation. Therefore, a surge of interest has been expressed lately in Internet security. Recent studies [1] indicate that botnets are the primary platform through which cyber criminals create global cooperative networks that are instrumental in most cyber criminal attacks. A bot is a software robot or a malware instance that runs autonomously and automatically on a compromised machine without being noticed by the victim user. The bot code is often written by skilled programmers and usually supports several kinds of malicious functionalities [2] that are instrumental in a variety of attacks and malicious activities. The term botnet, derived from the word bot, is a network of bots that are controlled by an attacker called a botmaster or botherder. The alarming increase in the power of botnets and its infectious effects have turned botnets into one of the biggest threats to the Internet security [3]. Currently, botnets are one of the root causes of most Internet attacks and malicious activities. Although the existence of botnets has been noticed for a long time, it is the recent growth of cyber crimes, which are mediated by botnets, that has attracted the attention of IT security researchers.

Each new generation of bots distinguishes itself by exploring different command and control (C&C) techniques, through which they can be updated and directed by the botmaster. One of the options employed is the Internet Relay Chat (IRC) protocol used by Agabot, SDBot, and SpyBot [2]. Some bots such as BlackEnergy [4], Rustock [5], and clickbot.A [6] rely on HTTP since it hardens the detection process because HTTP traffic is allowed in most network policies. Other types of botnets do not rely on centralized command and control mechanisms. Instead, they use distributed control techniques to avoid the single point of failure problem. For example, Storm [7], [8] and Nugache [9] use peer-to-peer networks to organize and control their network.

The primary objective of this paper is to conduct a reverse engineering study of the Zeus crimeware toolkit. The rationale underlying this exercise is two fold: First, as members of the National Cyber Forensics and Training Alliance (NCFTA) Canada, we frequently conduct reverse engineering and analysis of prominent malicious codes. The intent underlying such studies is to better understand malware inner workings, behaviors and enabling techniques and technologies. The insights gained from this will better position us to counteract the threats and contribute to cyber crime fight. Second, the choice of Zeus is the result of a discussion with our NCFTA partners. We reached the conclusion in June 2009 that Zeus was major threat that deserves a reverse engineering effort. In fact, this prediction was confirmed in July 2009 when a security publication from Damballa positioned Zeus as the number 1 botnet threat with 3.6 million infections in the US alone (about 19% of the installed base of PCs in the US [10]). It was also estimated that Zeus is guilty in 44% of the banking malware infections [11]. Recently, Symantec Corporation referred to this crimeware toolkit as the "King of the Underground Crimeware Toolkits" [12].

The Zeus crimeware toolkit has become one of the favorite tools for hackers because of its user friendly interface and its competitive price in the underground communities. This crimeware allows attackers to configure and create malicious binaries, which are mainly used to steal users' Internet banking

978-1-4244-7550-6/10/$26.00 ©2010 IEEE

accounts, credit cards, and other sensitive information that can be sold on the black market [13]. It also has the ability to administrate the collected stolen information through the use of a control panel, which is used to monitor, control, and manage the infected systems. To the best of our knowledge, there has been no reverse engineering attempt to de-obfuscate and analyze Zeus.

In this paper, we present a case study on the reverse engineering steps necessary to understand the inner working of the Zeus crimeware toolkit and its components. The main contributions of this paper are three folds. First, we present a detailed reverse engineering analysis of the Zeus crimeware toolkit to unveil its secrets and enable its mitigation. Second, we designed a tool to automat the recovery of the encryption key used for the bot communication and the extraction of the configuration information from the binary bot executables. This opens an opportunity to inject falsified information into the botnet communications which can be used to defame this crimeware toolkit. Third, we provide a breakdown for the structure of the Zeus botnet network messages.

The remainder of this paper is organized as follows. Section II is dedicated to the description of the Zeus crimeware toolkit components and how they are integrated. Section III details the network behavior analysis that is inferred from observing the network traffic between a bot instance and a the associated command and control server. In Section IV, we detail the four obfuscation levels and explain how they have been uncovered. This step led to the actual un-obfuscated code of the bot, and later to revealing the infection/installation process, and the encryption key that makes it possible to decrypt the C&C communications between the infected machine and the botnet infrastructure. We also present a sample decrypted communication session between an infected machine and a C&C server. Our conclusion is given in Section V.

II. Description of the Zeus Crimeware Toolkit

The Zeus crimeware toolkit is a set of programs which have been designed to setup a botnet over a high-scaled networked infrastructure. Generally, the Zeus botnet aims to make machines behave as spying agents with the intent of getting financial benefits. The Zeus malware has the ability to log inputs that are entered by the user as well as to capture and alter data that are displayed into web-pages [13]. Stolen data can contain email addresses, passwords, online banking accounts, credit card numbers, and transaction authentication numbers. In our analysis, we examine the Zeus crimeware toolkit v.1.2.4.2, which is considered as the latest stable publicly available version in the underground community. The overall structure of the Zeus crimeware toolkit consists of five components:

1) A control panel which contains a set of PHP scripts that are used to monitor the botnet and collect the stolen information into MySQL database and then display it to the botmaster. It also allows the botmaster to monitor, control, and manage bots that are registered within the botnet.
2) Configuration files that are used to customize the botnet parameters. It involves two files: the configuration file `config.txt` that lists the basic information, and the web injects file `webinjects.txt` that identifies the targeted websites and defines the content injection rules.
3) A generated encrypted configuration file `config.bin`, which holds an encrypted version of the configuration parameters of the botnet.
4) A generated malware binary file `bot.exe`, which is considered as the bot binary file that infects the victims' machines.
5) A builder program that generate two files: the encrypted configuration file `config.bin` and the malware (actual bot) binary file `bot.exe`.

On the C&C side, the crimeware toolkit has an easy way to setup the C&C server through an installation script that configures the database and the control panel. The database is used to store related information about the botnet and any updated reports from the bots. These updates contain stolen information that are gathered by the bots from the infected machines. The control panel provides a user friendly interface to display the content of the database as well as to communicate with the rest of the botnet using PHP scripts. The botnet configuration information is composed of two parts: a static part and a dynamic part. In addition, each Zeus instance keeps a set of targeted URLs that are fed by the web injects file `webinject.txt`. Instantly, Zeus targets these URLs to steal information and to modify the content of specific web pages before they get displayed on the user‘s screen. The attacker can define rules that are used to harvest a web form data. When a victim visits a targeted site, the bot steals the credentials that are entered by the victim. Afterward, it posts the encrypted information to a drop location that is meant to store the bot update reports. This server decrypts the stolen information and stores it into a database.

III. Zeus Botnet Network Analysis

In this section, we explain the network communication that occurs between the C&C server (the server containing the control panel) and an infected machine. Such analysis can be used to write IDS rules and anti-virus detection routines. In order to perform the network analysis, we built a sandbox environment to collect and analyze the network traces that are generated from the communication between the C&C server and one of the bot instances. We configured a web server, which act as the C&C server and the drop location. This server hosts all resources that are required to operate the botnet (`config.bin` file, PHP scripts and the MySQL database). To customize the malware, we used the builder program to generate the malware binary file which is configured to communicate with a C&C server. Within our environment, fake websites are generated to reflect real scenarios of botnet attacks. All necessary entries of the configuration file as well as the web injects scripts are modified to target the fake website. After infecting a machine with the bot binary file, we collected network traces for one day. During this session, the

user of the infected machine visited the targeted website and then used login credentials, personal information, and credit card information for testing purposes.

By analyzing the bot network communications, we can learn the overall behavior of the Zeus botnet. The network behavior of the Zeus botnet constitutes a starting point, where we can dig into the crimeware toolkit functionalities. Since the Zeus botnet is based on the HTTP protocol, it uses a pull-method to synchronize the botnet communications. From the collected network traces between a bot and a C&C server, we observe that the bot periodically checks specific server for an up-to-date configuration and bot binary files. Moreover, the HTTP communication messages between the two entities are encrypted. By observing the network trace, we managed to determine the following communication pattern between the C&C server and the infected machine:

1) The infected client starts the communication by sending a request message `GET /config.bin` to the C&C server. This message is a request to fetch the configuration file for the botnet.
2) The C&C server replies with the encrypted configuration file `config.bin`.
3) The client receives the encrypted configuration file and decrypts its content by using an encryption key, which is embedded inside the bot binary file.
4) Situation where, the botmaster wants to involve the infected machine to manage the botnet, the infected machine has to provide its external IP address and report any use of Network Address Translation (NAT). In order to know the external IP address that is seen by the botnet servers, the infected machine makes a request to a specific server. Afterward, this server informs the infected machine about their externally facing IP address. The server's URL is provided in the static configuration file.
5) The bot posts the stolen information and its update status reports to the C&C server `POST/gate.php`.

Figure 1 illustrates the communication pattern between the C&C server and the infected machine. The communication pattern is repeated frequently depending on a timing variable, which is defined in the botnet configuration file.

IV. Reverse Engineering Analysis

The increasing usage of malicious software has pushed security experts to try to find the secrets related to the development of malware design. A common technique to detect the existence of a given malware is by tracking system modifications. The changes include what an operating system runs at startup, changes of default web pages, generated traffic, infection of processes, packing/unpacking of binaries, and changes to the registry keys. One way to look for these changes is to reverse engineer the malware and try to reveal what is hidden behind the assembled code. In our case, this kind of analysis provides an invaluable insight into the inner-working of the crimeware toolkit in general and about the malware binary in particular. In the stream of this thinking, we investigate the builder program and malware binary file.

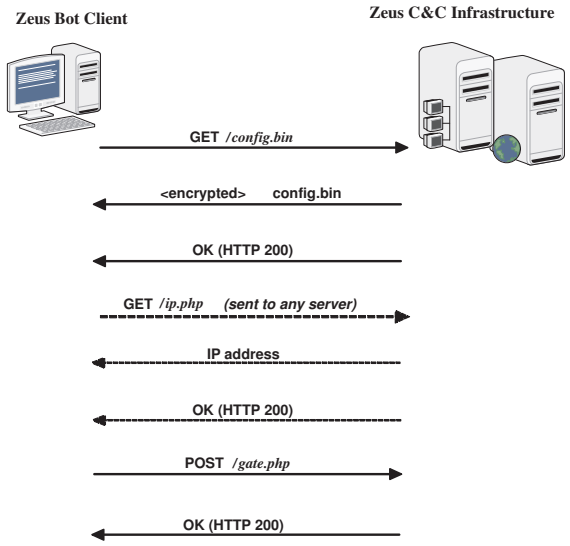

Fig. 1. Communications pattern of Zeus

To this end, we mainly employ "IDA Pro" [14] to disassemble the binaries and debug them to understand their business logic. The analysis is two folds: First, the analysis that is related to the builder program. Second, the analysis that is linked to the malware binary file.

A. The Zeus Builder Program Analysis

The builder is one of the components of the Zeus crimeware toolkit. It uses the configuration files as an input to generate the bot binary file and the encrypted configuration file.

We analyze the builder program first because it uses a known obfuscation technique that can be easily removed. In addition, the GUI allows us to categorize different subroutines, which make up the builder program functionalities. Using the "PaiMei" reverse engineering framework [15] (which is a reverse engineering framework that provides many reverse engineering tasks such as fuzzer assistance, code coverage tracking, and data flow tracking), we were able to see exactly what functions of the builder program are invoked by a specific action. This immensely aids in simplifying the reverse engineering efforts as it allows us to focus on a few key subroutines at a time. In the following, we summarize the reverse engineering analysis of the functionalities of the builder program.

Building the Configuration File Functionality
: This function is responsible for encoding the clear text of the configuration files of the botnet into a specific structure. Afterwards, it encrypts the whole structure with the RC4 encryption algorithm using the configured encryption key.

Building the Malware Binary File Functionality
: The main function of the builder program resides within this functionality, which is responsible for building the customized malware binary files. In general, it builds the malware executable file into a

portable executable (PE) standard format. Moreover, it sets some parameters according to the current configuration file and then produces the malware binary file.

Malware Infection Removal Functionality

The builder has a functionality that ascertains the presence of Zeus bot and removes it. When this functionality runs, it performs a detection routine by checking the existence of special registry keys that are inserted during the bot infection process. Also, it detects the presence of some files in the system. If these files are detected, the builder program cleans some registry keys and instructs the bot to shutdown itself and then deletes the stored Zeus binary file from the system.

The expected behavior of the bot when it receives the shutdown command is to disinfect itself from the currently running processes. The analysis reveals the name of files that the builder checks their presence in the system. Table I represents these file names with their description.

B. Zeus Bot Binary Analysis

As depicted in Figure 2, the bot binary file contains four segments: a "text/code" segment, an "imports" segment, a "resources" segment, and a "data" segment. Therefore, we begin our analysis at the malware Entry Point (EP) that resides in the "text/code" segment. The initial analysis of the disassembly reveals that only a small part of the "text/code" block is valid computer instructions. The rest of the binary is highly obfuscated, which means that the computer cannot use these segments directly unless it is de-obfuscated at some stage.

1) De-obfuscation Process: Using the "IDA Pro" debugger, we were able to debug the malware and step through the instructions to analyze and understand the logic of the de-obfuscation routines. Each routine reveals some information which is used by the other routines until all obfuscation layers are removed. The first de-obfuscation routine contains a 4-byte long decryption key and a one-byte long seed value. These two values are used to decrypt a block of data from the "text/code" segment and then write the decrypted data in the virtual memory. The result of the first de-obfuscation routine revealed some new code segments. These segments contain three de-obfuscation routines as shown in Figure 3. During our analysis, the initial offset address of the memory for the code segments was `0x390000`. After the address space of the second de-obfuscation routine, there was an 8-byte key that the "IDA Pro" incorrectly identified as code instructions. Figure 4 illustrates the location of the 8-byte key. In the following, we explain the main logic of the second de-obfuscation routine.

1) First, it copies two binary blocks from the "text/code" segment, concatenates them together, and then writes them into the virtual memory. The first text block contains data with many zero value bytes that will be filled by the next text block as shown in Figure 5.

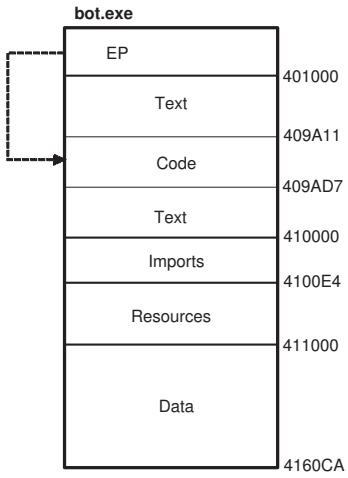

Fig. 2. Segments of the `bot.exe` binary file

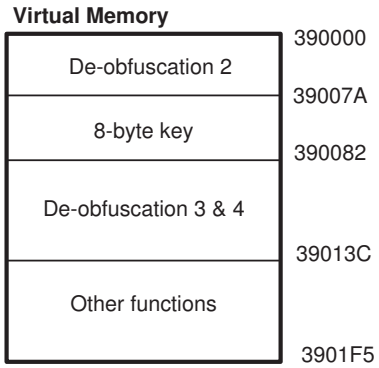

Fig. 3. De-obfuscated code in the virtual memory

2) The routine scans every byte on the first text block and when it encounters a "hole" (zero byte), it will overwrite the zero byte with the next available byte in the "filler" text block. This is repeated until all "holes" are filled (See Figure 6).

The filled text segment turns to be the main outcome of the second de-obfuscation routine. However, this text segment is still not readable and not considered as computer instructions. By utilizing the 8-byte key, the third de-obfuscation routine starts by decrypting the output of the second de-obfuscation. Similar to the first de-obfuscation routine, this routine utilizes the 8-byte key and performs an eXclusive-OR (XOR) operation instead of an addition operation. Finally, the fourth de-obfuscation layer contains heavy computations to initialize and prepare some parameters for the rest of the malware operations. It uses the decrypted bytes revealed by the previous routines to modify the rest of the "text/code" segment. After this routine completes, we can observe the real starting point of the Zeus malware. Even though the "text/code" segment is now valid, the Zeus bot binary employs two additional layers

File	Description
C:/WINDOWS/system32/sdra64.exe	A copy of a bot which has infected "system32" folder.
C:/WINDOWS/system32/lowsec/local.ds	A data storage file which is used to store the configuration file that is used by a given bot locally in the system.
C:/WINDOWS/system32/lowsec/user.ds	A data storage file which is used to log the users' activities that have been recorded by the bot.

TABLE I
DESCRIPTION OF THE FILES THAT ARE CREATED DURING THE BOT INFECTION

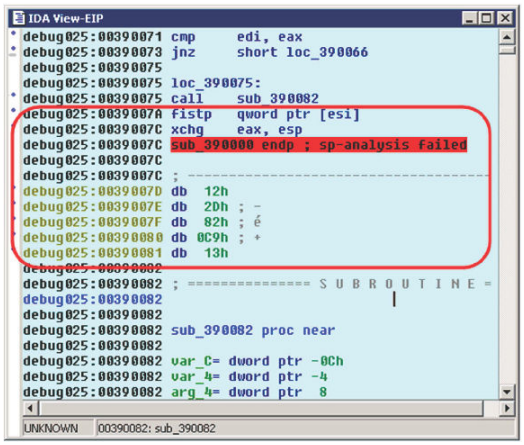

Fig. 4. The 8-byte key

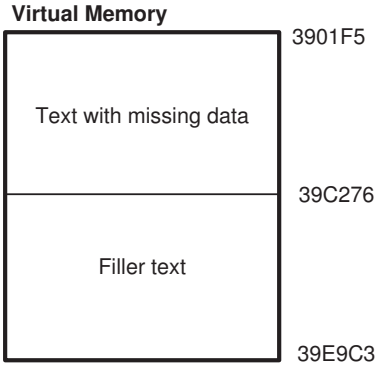

Fig. 5. The virtual memory used by the second de-obfuscation routine

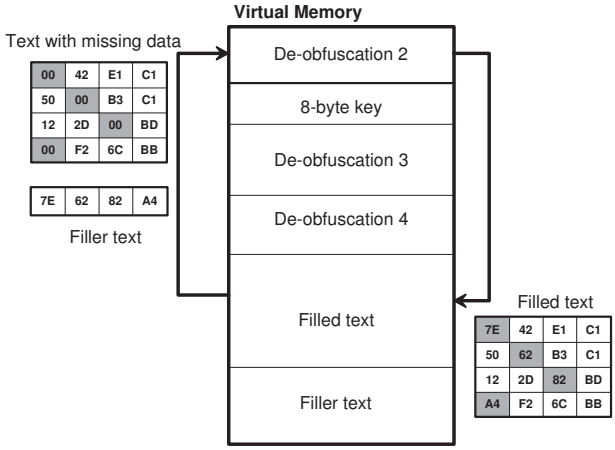

Fig. 6. The result from the second de-obfuscation routine

Algorithm IV.1: DECRYPT_STRING(enc_string)

```
seed = 0xBA;
String new_string = new String(enc_string.length());
for i = 0 to enc_string.length()
  do { new_string[i] = (enc_string[i] + seed) %256;
       seed = (seed + 2);
return (new_string)
```

Algorithm IV.2: DECRYPT_URL(enc_url)

```
String new_url = new String(enc_url.length());
for i = 0 to enc_url.length()
  do { if (i%2 == 0)
         then
       new_url[i] = (enc_url[i] + 0xF6 - i * 2) %256;
       else
       new_url[i] = (enc_url[i] + 0x7 + i * 2)%256;
return (new_url)
```

of obfuscation. These two layers are de-obfuscated during the installation procedure. They consist of logical loops that transform arbitrarily long strings into a readable text. The first layer is performed on a set of strings that the malware uses to load the DLL libraries, retrieve function names, and for other purposes during the installation process. Similarly, the second layer is used to decrypt URLs in the static configuration of the configuration file. The main logic of these two routines are described in Algorithm IV.1 and Algorithm IV.2.

2) Bot Installation Process: After the first four de-obfuscation routines are executed, the malware begins the installation process. The installation process aims at preparing and then launching the malicious activities of the malware. In the following, we explain the main procedure of the installation process.

1) The Zeus malware dynamically loads the `LoadLibrary` and the `GetProcAddress` methods from `Kernel32.dll` library.
2) It decrypts the set of strings, which become DLL methods names, into the virtual memory according to Algorithm IV.1.
3) The `LoadLibrary` and the `GetProcAddress` methods are then used to load the further methods, as decrypted in step 2, from the Windows DLLs.
4) The Zeus malware enumerates the current process table looking for targeted processes such as the main process name for the Outpost personal firewall application from Agnitum Security `outpost.exe` and the main process name for the personal firewall of the ZoneLabs Internet security `zlclient.exe`. If any of these processes is found, then the Zeus malware aborts the installation process.
5) The Zeus malware appends the path `C:/Windows/System32/sdra64.exe` to `HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/WindowsNT/CurrentVersion/Winlogon/Userinit` registry key. This entry enables the Zeus malware to initiate its installation process again during Windows startup.
6) Finally, it injects its entire Zeus binary file from the memory address `0x400000` to `0x417000` into the virtual memory of `winlogon.exe` process. After that, Zeus passes the control to this process by creating a new user thread, which is immediately executed.

Similarly, the bot uses these steps when the infected machine is restarted. However, there are few steps that are performed only during the initial Zeus installation process. These steps are related to the creation of a local copy of the malware and storing it on the infected system for further activities. In the following, we list the main process of creating a local copy of the malware.

(a) The Zeus malware searches for any existing copies of previous Zeus infection files `sdra64.exe`, and then erases it from the infected machine. This behavior would occur when the Zeus binary file is being updated with a newer version of the malware.
(b) It makes an exact copy of itself and then saves it to `C:/Windows/System32/sdra64.exe`. To evade signature-based detection systems, it appends some randomly generated bytes to the end of the file.
(c) In order to hide itself, the bot duplicates the Modification, Access, and Creation times (MAC times) information from `Ntdll.dll` library, and applies them to the `sdra64.exe`. The intent of this is to make `sdra64.exe` appears to be a system file that has been around since Windows was first installed.
(d) In another level of hiding the created file, it sets the `sdra64.exe` file attributes to system and hidden, so that the user cannot see the file using the standard file explorer.

At this stage, the malware is already injected within the `winlogon.exe` running process. On the other hand, the currently running bot exits and leaves the control to the injected process. However the installation procedure is continued by the user thread that was started in the `winlogon.exe` process as described in step 6. From the injection process, we infer that the entire Zeus binary file is copied into the `winlogon.exe` process. Therefore, the injected Zeus instance starts by removing the remaining two layers of the obfuscation by applying Algorithm IV.1 and Algorithm IV.2 as described in Section IV-B1. When the injected malware decrypts all the strings, the Zeus instance employs the piggyback thread technique (to control the infected system through legitimate process) within the `winlogon.exe` process. However, Zeus instances only perform few tasks before they create another thread and exit themselves. This is another attempt by the designers of the Zeus malware to evade detection. Afterwards, the Zeus instance starts injecting itself into another process, namely the `svchost.exe` process. This injected process initiates a communication channel with the C&C server to download the latest updates on the configuration file and the malware itself. Later, the targeted processes get injected with the latest malware payload and then activate the process of stealing information through API hooking techniques. During the malware update process, the following changes were observed on the file system:

1) A new folder is created at the path `C:/Windows/System32/lowsec`. Hiding techniques similar to these that are applied to the `sdra64.exe` are also applied to the created folder.
2) Two new files, `local.ds` and `user.ds`, are created and placed in the new created folder. The `user.ds` stores the dynamic configuration file, and the `local.ds` logs the stolen information until the Zeus malware is ready to send it to the drop location.

The malware that resides on the `winlogon.exe` process acts as the brains for the Zeus malware activities. It communicates and coordinates all the infected process using the named pipe `_AVIRA_2109`. Table II shows the list of the commands that are supported by the Zeus malware.

3) Key Extraction: As mentioned in Section II, the Zeus botnet uses a configuration file that contains a static information. Specifically, this part of the configuration is stored inside the malware binary file in a specific structure. During the de-obfuscation processes, this structure is recovered and placed in the virtual memory (In our analysis, starting at `0x416000`). All information in the structure is completely de-obfuscated except for two URLs: `url_compip` and `url_config`. These URLs can be de-obfuscated using Algorithm IV.2. The `url_compip` is the web location to determine the IP address of the infected host, and the `url_config` is the web location to download the configuration file for the botnet. The static configuration structure also contains an RC4 substitution table that is generated by the encryption key specified in the configuration file. Throughout our analysis, we noticed that the substitution table were generated by the RC4s key-scheduling algorithm and then we verified that the encryption employed

Command	Purpose	Return Value
1	Retrieve Zeus version number	4 bytes in a buffer
2	Retrieve name of the botnet	Ascii string in buffer
3	Uninstall Bot	n/a
4	Open the `local.ds` file or create it if it does not exist	n/a
5	Close the `local.ds` file	n/a
6	Open the `user.ds` or create it if it does not exist	n/a
7	Close the `user.ds`	n/a
8	Close the `sdra64.exe`	n/a
9	Open the `sdra64.ex`	n/a
10	Retrieve loader file path	Wide character string
11	Retrieve configuration file path	Wide character string
12	Retrieve log file path	Wide character string
13	Crash the `winlogon` process intentionally	n/a

TABLE II
LIST OF THE ZEUS MALWARE COMMANDS

by Zeus is done by the RC4 algorithm. The recovered static configuration can be used in different ways to gain some control over the botnet. The most valuable piece of information is the substitution table which can be used to decrypt all the communications of the Zeus botnet. Moreover, it can be used to decrypt the configuration file as well as the stolen information. In order to recover the static configuration structure described above, we have to go through all the de-obfuscation phases discussed in Section IV-B1. This requires executing the malware until it finishes all the de-obfuscation layers. Emulation techniques are considered as a safe and fast procedures to achieve our goals. Using Python scripting language along with the "IDAPython" plugin [16], we were able to emulate all the de-obfuscation routines and extract the substitution table from the static configuration structure. These extracted keys allows for decrypting the botnet communication traffic and all the encrypted files. Similarly, it allows us to extract any information from the static configuration structure, such as the URLs for any future updates, which point to the C&C servers. Our experimental results show that any subversion of Zeus (v.1.2.x.x) can be fully analyzed using our methodology because it holds the same logical blocks.

C. Packet Decryption

After extracting the RC4 encryption key as described in Section IV-B3, we used it to decrypt the botnet communications. By decrypting the transmitted HTTP payload, we are able to uncover the structure of the messages between the bot and the C&C server. We analyzed the structure of the HTTP POST messages (`POST /gate.php`) which carries all the updates and reports from the bots to the C&C server. Each bot posts a variable number of encrypted bytes based on the sent data to the C&C server in a specific structure. The payload is encrypted using an RC4 encryption algorithm only. As depicted in Figure 7, we restore the structure of the messages as follows:

1) Each message starts with a header that consists of 28-bytes. This header contains an MD5 hash value for the rest of the message.
2) As shown in Figure 7, the rest of the message follows in the form of repeated data blocks where each block consists of:
   a) An entry header with 16-bytes that contains information about the current data entry. The first 4-bytes serve as the type of the reported information, which can be recognized by the bot and the control panel. The third 4-bytes determine the length of the carried information.
   b) A variable number of bytes that is specified in the entry header. These bytes represent one piece of the information that is transmitted within this packet.

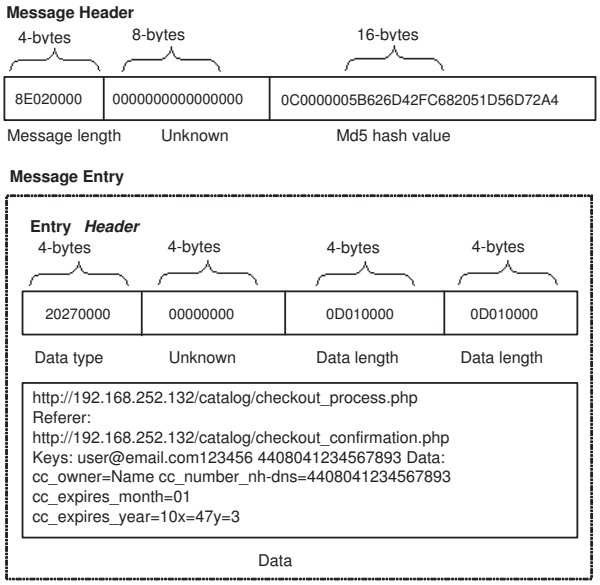

Fig. 7. A decrypted sample message

It should be noted that the encrypted communication of the Zeus botnet is vulnerable to the RC4 keystream reuse attack because there is no Initialization Vector (IV) setup in every session, i.e., the same RC4 keystream is reused to encrypt all messages.

V. Conclusion

The Zeus crimeware toolkit is an advanced tool used to generate very effective malware that facilitates criminal activities. The integrated tootkit technology harden the detection of the malware at the host level. Similarly, the use of encrypted HTTP messages for C&C makes it difficult to detect any clear behavior at the network level. Moreover, the multiple levels of malware obfuscation presents a burden in front the analysts to find information about the C&C servers or to generate binary signatures. In this work, we presented a detailed reverse engineering analysis of the Zeus crimeware toolkit to unveil its underlying architecture and enable its mitigation. We have also designed a tool to automat the recovery of the encryption key and the extraction of the configuration information from the binary bot executables. Furthermore, we provided a breakdown for the structure of the Zeus botnet network messages.

Our analysis of the C&C communications indicates that the RC4 algorithm is used in a poor way to encrypt these communications (keystream reuse). In addition to the knowledge of the network messages structure, we can launch an active countermeasures by interacting with the botnets servers using the extracted encryption key. For example, we can inject falsified information into the botnet communications for various purposes, such as defaming the botnet business model by reducing the effectiveness of their services [17], [18]. A useful extension to our work is to use the extracted encryption key mechanism in order to analyze and track down the Zeus C&C servers or to defame the toolkit, e.g., by returning fake (invalid) credit card numbers.

References

[1] L. Wenke, W. Cliff, and D. David, Eds., *Botnet Detection: Countering the Largest Security Threat*, ser. Advances in Information Security. Springer-Verlag New York, 2008, vol. 36.

[2] P. Barford and V. Yegneswaran, "An inside look at botnets," *Malware Detection*, pp. 171–191, 2007.

[3] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, pp. 291–302, 2006.

[4] J. Nazario, "Blackenergy DDoS bot analysis," Arbor Networks, Tech. Rep., 2007.

[5] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007.

[6] N. Daswani and M. Stoppelman, "The anatomy of clickbot.A," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007.

[7] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-Peer botnets: overview and case study," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007.

[8] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of Peer-to-Peer based botnets: a case study on storm worm," in *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. Berkeley, CA, USA: USENIX Association, 2008, pp. 1–9.

[9] D. Dittrich and S. Dietrich, "P2P as botnet command and control: a deeper insight," in *3rd International Conference on Malicious and Unwanted Software (MALWARE)*, Appl. Phys. Lab., Univ. of Washington, Washington, DC, USA. Piscataway, NJ, USA: IEEE, 7-8 Oct. 2008 2008, pp. 41–48.

[10] Top-10 botnet outbreaks in 2009. [Online]. Available: http://blog.damballa.com/?p=569

[11] Banking malware zeus sucessfully bypasses anti-virus detection. [Online]. Available: http://www.ecommerce-journal.com/news/18221_zeus_increasingly_avoids_pcs_detection

[12] Zeus, king of the underground crimeware toolkits. Symantec Corporation. [Online]. Available: http://www.symantec.com/connect/blogs/zeus-king-underground-crimeware-toolkits

[13] T. Holz, M. Engelberth, and F. Freiling, "Learning more about the underground economy: A case-study of keyloggers and dropzones," *Computer Security ESORICS 2009*, pp. 1–18, 2009.

[14] IDAPro - Multi-processor disassembler and debugger. [Online]. Available: http://www.hex-rays.com/idapro/

[15] PaiMei - a reverse engineering framework. [Online]. Available: http://code.google.com/p/paimei/

[16] IDAPython: an IDA Pro plugin. [Online]. Available: http://d-dome.net/idapython/

[17] Z. Li, Q. Liao, and A. Striegel, "Botnet economics: Uncertainty matters," *Managing Information Risk and the Economics of Security*, pp. 245–267, 2009.

[18] R. Ford and S. Gordon, "Cent, five cent, ten cent, dollar: hitting botnets where it really hurts," in *NSPW '06: Proceedings of the 2006 workshop on New security paradigms*. New York, NY, USA: ACM, 2007, pp. 3–10.