A Peer to Peer Botnet: Implementation, Detection and Mitigation

[1]Sanket N. Patel, 2Tarulata Chauhan

Department Of Computer Engineering
L.J. Institute of Engineering & Technology, Ahmedabad-382210, India
[1]patel.sanket.sanket@gmail.com

***Abstract*--In this paper, we present our Implementation, Detection and Mitigation results for Zeus botnet. This is one of the recent and powerful crime ware botnet that emerged in the Internet underground community to control botnets. Our implementation and analysis aims at understanding Zeus botnet working scenario based on that we detect the Zeus communication and also try to uncovering the various obfuscation levels and shedding the light on the resulting code. In addition, we detail a method of implementation also detect the behavior based on network analysis and try to decrypt the network communication and the botnet configuration information.**

1. INTRODUCTION

In recent years, cyber attacks have evolved, becoming more profit-centered and better organized. Emails spams, click frauds, and social phishing are becoming more and more popular. Botnet, which consists of a network of compromised computers connecting to the Internet and controlled by a remote attacker, are for all of these problems. A botmaster can drive numerous compromised computers to attack the target at the same time. Therefore, as an attack platform, a botnet can cause serious damage and is very hard to defend against.

Compare to other attack vectors, the essential component of a botnet is its control and command channel. A C&C is used to update the bot program, distribute commands from the botmasters, and collect victims' private information, such as bank accounts and identifying information. Once a C&C is broken down, the botnet degrades to discrete, unorganized infections, which are easy to be eliminated using host-based cleanup technology.

The Botnet Lifecycles [1]

1. **Infection**: initial installation of botnet malware on target host. this is done by tricking users into running executables to attached to email. By exploiting their browser weakness or exploiting the presence of security holes.
2. **Bootstrapping and Maintenance**: each node has to perform a set of actions to detect the presence of other nodes and connect to them, bot controller must be able to counteract when its associated nodes leave the botnets. Such maintenance operation has a fundamental role in ensuring robustness.
3. **Command & Control**: botmaster and controller have the necessity of reliably distributing their command(e.g by sending the command start ddos target=192.133.0.10 or send spam mail templates bot software updates) to their controlled nodes. that in turn to send associated results, or current status into back to bot master.
4. **Command Execution**: running the received command on each individual bot.

Command and Control Architecture [2]

A second core problem for botnet attackers is how to communicate with each bot instance. Most attackers would like the ability to rapidly send instructions to bots but also do not want that communication to be detected or the source of the those commands to be revealed.

1. **Centralized**: A centralized topology is chara- cterized by a central point that forwards messages between clients. Messages sent in a centralized system tend to have low latency as they only need to transit a few well-known hops. From the perspective of an attacker, centralized systems have two major weaknesses: they can be easier to detect since many clients connect the same point, and the discovery of the central location can compromise the whole system.
2. **P2P**: Peer-to-peer (P2P) botnet communication has several important advantages over centralized networks. First, a P2P communication system is much harder to disrupt. This means that the compromise of a single bot does not necessarily mean the loss of the entire botnet. However, the design of P2P systems is more complex and there are typically no guarantees on message delivery or latency.
3. **Unstructured**: A botnet communication system could also take the P2P concept to the extreme and be based on the principle that no single bot would know about any more than one other bot. In such, a topology a bot or controller that wanted to send a message would encrypt it and then randomly scan the Internet and pass along the message when it detected another bot. The design of such a system would be relatively simple and the detection of a single bot would never compromise the full botnet. However, the message latency would be extremely high, with no guarantee of delivery.

The Zeus crime ware botnet has become one of the favorite tools for hackers because of its user friendly interface. This crime ware allows attackers to configure and create malicious binaries, which are mainly used to steal users Internet banking accounts, credit cards and other sensitive information that can be sold on the black market[zeus valu paper].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 system. To the best of our knowledge, there has been no reverse engineering attempt to de–obfuscate and analyze Zeus.
The remainder of this paper is organized as follows. Section 2 is dedicated to types of botnet attacks. Section 3 details the implementation of port scanning analysis Ourmon tool. Section 4 details the implementation of Zeus Crime ware. In Section 5, we detail detection of Zeus by wire shark. Section 6 details the decompile of Zeus executable binary. Our Conclusion is given in Section 7.

2. BOTNET ATTACKS.[3]

1. **Attacking IRC Networks**: Botnets are used for attacking IRC networks. The victim is flooded by service request from thousands of Bots and thus victim IRC network is brought down.
2. **Distributed Denial of Services (DDoS)**: DDoS is a attack on a computer system or network that causes a loss of services/network to users by consuming the bandwidth of the victim network. The resource path is exhausted if the DDoS- attack causes many packets per second (PPS). The DDoS attacks are not limited to Web servers, virtually any service available on the internet can be target of such an attack. Higher level protocols can be misused to increase the load even more effectively by using very specific attacks such as such as running exhausting search queries on bulletin boards or recursive HTTP-floods on the victim's website called spidering.
3. **Key Logging**: With the help of a key logger it is very easy for an attacker to retrieve sensitive information. There exists filtering mechanism that aid in stealing secret data.
4. **Sniffing Traffic**: Bots can also use a packet sniffer to watch for clear text data passing by compromised machine. The sniffers are used to retrieve sensitive information such as usernames and passwords.
5. **Spamming**: Some bots can open a SOCKS v4/v5 proxy—a generic proxy protocol for TCP/IP-based networking applications—on a compromised machine. After having enabled the SOCKS proxy, this machine can then be used for nefarious tasks such as spamming. With the aid of Botnet, an attacker can then send massive amounts of bulk e- mail (spam). Some Bots also harvest e-mail addresses (by opening a SOCKS v4/v5 proxy).
6. **Advertisement Installation**: BotNets setup a fake web site with some advertisements. The operator of this website negotiates a deal with some hosting companies that pay for clicks on ads. With the help of Botnet, these clicks can be 'automated' so that instantly a few thousands Bots clicks on the pop-ups, hijacks the start page of a compromise machine so that the 'clicks' are executed each time the victim uses the browser.
7. **Spreading New Malware**: This is easy since all Bots implement mechanisms to download and execute a file via HTTP or FTP. Thus, spreading virus via e-mail is very easy using a Botnet.
8. **Manipulating Online Polls or Games**: These are very easy to manipulate due to high attention. Since every Bot has a distinct IP address and do the manipulation. Every vote will have the same credibility as a vote cast by a real person. Online games can be manipulated in a similar way.
9. **Mass Identity Theft**: By combining above different functions, they are used for large scale identity theft which is one of the fastest growing crimes on the internet. Bogus e- mails that pretend to be legitimate (such as banking e-mail) ask their internet victims to go on line and submit their private information. The fake e-mail are generated and sent by Bots via their spamming mechanism. These Bots can also host multiple fake web sites and as and when one of these fake sites is shut down, another one can pop up. In addition, key logging and sniffing of traffic can also be used for identity theft.

3. IMPLEMENTATION OF OURMON FOR DETECTION OF SCANNING

Basically the ourmon detection tool is network anomaly tool it concentrate on abnormal behavior of traffic. For peer to peer and scanning detection two part is important TCP anomaly Detection and UDP anomaly Detection.

TCP anomaly Detection

This is the TCP port report in several forms and formats. First we look at the basic 30-second ASCII TCP port report. The port report is useful for detecting scanning and P2P activity. Second we look at the TCP work weight, which is a statistical measure that is mostly used to detect scanning. The TCP work weight is a fundamental background component for all TCP-based anomaly detection.

Ip_src	Flags	Apps	Work	SA/S	L3D/L4D	L4S/src	Snt/rcv	Port Signature
10.0.0.1	WOR		100	0	41/1	10/3441	85/28	[5900,100]
10.10.10.10	OR	H	17	100	3/26	1/80	124/147	[2829,10]***
10.59.153.150	EWOM	P	100	100	53/1	10/1069	54/0	[445,100]
192.168.153.150	W	P	94	0	379/4	10/8338	784/34	[139,23][445,65]***
192.168.153.151	Ew	I	81	0	3/26	10/2334	624/44	[139,15][445,60]***
192.168.160.1		G	13	0	193/155	10/8339	1k/1k	[1256,9][6346,43]***

Fig.1.TCP port report.

The fundamental object in the TCP port report is an IP host address and its associated statistics. In particular, the 30-seconds version of the TCP port report is sorted by ascending IP address.

Note how the port signatures for 192.168.153.150 and 192.168.153.151 match; this isn't an accident. They are running the same malware that is currently performing same scan on both hosts.

- **10.0.0.1 -** The *R* flag indicates RESETS are coming back. The work weight is 100 percent.L3D/L4D indicates this host is talking to many local hosts at only one port. One destination port is the target (port 5900).This is a scanner, plain and simple. At this point if you don't know what is going on, use a search engine and search on *TCP port 5900*. In this case we can rapidly learn that port 5900 is associated with a the Virtual Network Connection (VNC) application, and some version of it must have a bug as a hacker or a bot is looking for hosts to attack using a VNC exploit. Another possibility is that it might be used on hacker boxes and represents some sort of backdoor port. The network authorities might want to make sure port 5900 is protected in some manner.

- **10.10.10.10 -** Here we have a false positive, most likely. The H flag means a Web source port was seen, and sure enough, L3S/src shows one source port, port 80. SA/S is also 100 percent, which indicates a likely server. The port signature itself has random high ports in it which suggests dynamically allocated client ports. Web servers sometimes do show up in the basic port report. Of course, the strongest thing we can say here is that the work weight itself was only 17 percent. Therefore it is low and not worrisome. The former, when nonzero, can indicate hosts with multithreaded applications that open multiple threads for efficiency but unfortunately have a high ratio of TCP control packets to data packets (this includes Web servers and P2P clients on hosts). If the number is above 70 percent for several instances of the TCP port report, you probably have a scanner, although it is always possible to have a client that has some sort of problem (like no server).We will say more about false positives in a moment. This is a Web server.

- **10.59.153.150 -** Here we instantly know that we have a bad one. Why? Because it has a P for the application flags, meaning that it is sending packets into our darknet (honeypot). EWOM flags indicate (especially M) that packets aren't coming back. One way TCP is not how TCP was intended to work (TCP is for dialogues, not monologues). Interestingly enough, we also have 100 percent for the work weight and 100 percent for the SA/S value. This tells us the interesting and curious fact that more or less all the packets being sent are SYN+ACK packets. Some scanning uses SYN+ACK packets to get around older IDS systems that only detect SYN packets but assumed SYN+ACK packets came from TCP servers. Note that port 445 is the target (which is often the case). This is a scanner and could easily be part of a botnet mesh, too.

- **192.168.153.150 -** This IP and the next IP are local and are on the same subnet. As it turns out, both of these hosts belong to Botnet Client."These two hosts are infected with a botnet client and have been remotely ordered via the IRC connection to scan for exploits. Sure enough, the port signature shows that a large percentage of the packets on those hosts are being directed toward ports 139 and 445. 192.168.153.150 has a P, so it has been scanning into the darknet. Its work weight is 94 percent, too, which is too high.

- **192.168.153.151 -** 192.168.153.151 is scanning in parallel with the previous host. It is possible that one of these two hosts infected the other host. In this case the application flag has an I, which indicates IRC. This is often not an accident with a scanning and attacking host. It indirectly indicates the IRC channel used for controlling the botnet. Of course, IRC is often used for benign reasons, too, but not in this case.

- **192.168.160.1 -** Our last host is another example of a possible false positive. Here we have a host that is using a Gnutella application of some sort. The G in application flags indicates Gnutella. The work weight is low here, although Gnutella can have high work weights at times. The L3D/L4D values are very common for P2P using hosts because they are both high. In some sense this is the definition of peer to peer. A host talks to many other hosts (IP destinations) at possibly many TCP destination ports. The snt/rcv value is also interesting as it is both high and evenly distributed between packets sent and received. We say P2P hosts may be a false positive, but they might be what you wanted to catch anyway. This depends on whether the local security policy allows P2P or not.

TCP Work Weight: Details[4]

First of all, let's look at how the work weight is computed. The rough equation for the work weight for one IP host is:

$$\text{TCP work weight} = \frac{SS+FS+RR}{TP}$$

Where:

- SS is the total number of SYNS sent by the IP during the sample period.
- FS is the total number of FINS sent by the IP during the sample period.
- RR is the total number of TCP RESETS returned to the IP during the sample period.
- TP is the total number of TCP packets, including control and data sent and received by the host, during the sample period.

Roughly one easy way to understand this is that we are comparing the number of control packets to the count of all packets sent. If it is 100 percent, that means all control packets were sent, which means either the client/server TCP protocol is broken or somebody is doing some sort of scan.

UDP Anomaly Detection

The famous SQL-slammer was such a case; it contained a complete machine program in one UDP packet payload that exploited a SQL server and created a fearsome Internet wide flash storm in just a few minutes.

Here we are going to briefly look at two ourmon facilities for watching for UDP anomalies. The first is the UDP port report, which, like the TCP port report, is collected every 30 seconds. The second UDP facility is the RRDTOOL based UDP weight graph.

Ip src:	Weight:	Udp_ sent:	Udp_ recv:	Unreachs:	L3D/L4D	Appflags:	Port_sig
10.16.208.23	38386361	88261	0	2293	4322/2	Ps	[1025,50] [1026,50]

Fig.2.UDP port report

The UDP weight graphs a metric called the UDP work weight. So as with TCP and its port report, there is also a UDP port report and per IP host UDP work weight. In the UDP port report, for each UDP host address we compute a UDP work weight based on a 30-second packet count. The work weight is computed more or less as follows:

UDP ww = UDP packets sent * ICMP errors returned

If a host sends a lot of UDP packets fast and they cause common ICMP errors like destination host unreachable a high UDP work weight will be earned. Informally this means that the Internet found what you were doing to be in error. These sorts of events are often associated with DOS attacks.

Given that our normal top entry in the UDP port report has an average work weight of less than 100000, this one does seem to be interesting. The UDP work weight is around 380 million. So the aggressor sent 88k UDP packets and none were returned during the sample period. However, it got back about 2k UDP errors. Earlier we oversimplified our UDP work weight compute equation. We actually weight the ICMP errors in such a way that if a host receives ICMP errors, it will get a higher work weight. We show unique IP destination and UDP port destination counts as with the TCP port report. This shows that the host sent packets to 4k local hosts (a lot) at only two ports. It's clearly a scanner of some sort. We also have a few application flags (not many). P means that packets were sent into the darknet, and s is a built-in ourmon signature for identification of some forms of SPIM. Our port signature mechanism is completely the same as with the TCP port report. Here we see that half the UDP packets were sent to port 1025 and the other half were sent to port 1026.

4. IMPLEMENTATION OF ZEUS CRIME WARE

The Zeus crime ware 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 [5]. Stolen data can contain email addresses, passwords, online banking accounts, credit card numbers, and transaction authentication numbers. The overall structure of the Zeus crime ware 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(Fig.3).It also allows the botmaster to monitor, control, and manage bots that are registered within the botnet.

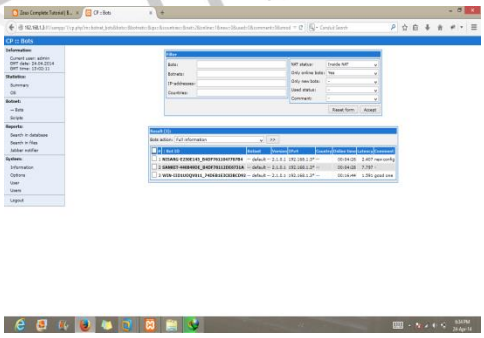

Fig.3. Zeus Control Panel Setup.

2. Configuration files that are used to customize the botnet parameters. It involves two files: the configuration file config.txt(Fig.4) that lists the basic information and the web injects file webinjects.txt(Fig.5) that identifies the targeted websites and defines the content injection rules.

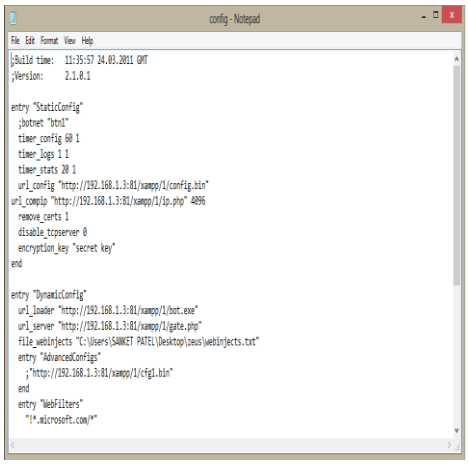

Fig.4. Config.txt customized parameters.

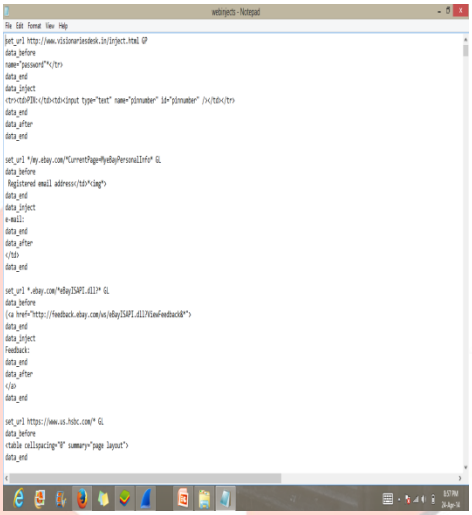

Fig.5. webinject.txt injected parameters.

3. A generated encrypted configuration files 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 machine.
5. A builder program that generates two files: the encrypted configuration file config.bin(Fig.6) and the malware binary file bot.exe(Fig.7).

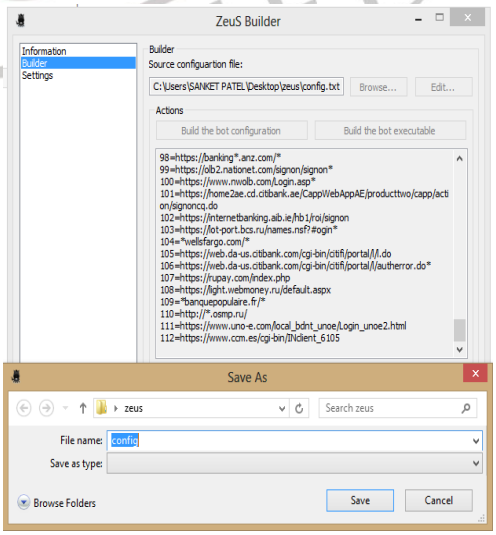

Fig.6. building of config.bin using Zeus Builder.

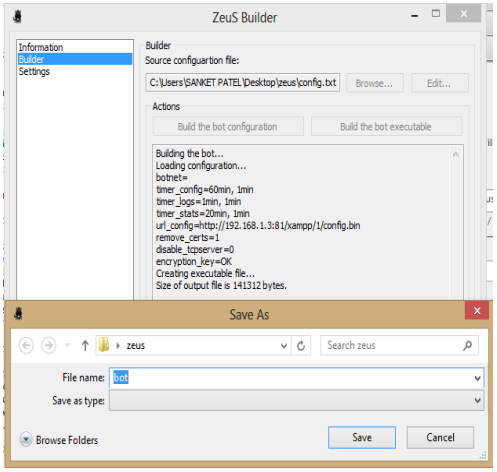

Fig.7. building of bot.exe using Zeus Builder.

Now here the botmaster IP address is configured in config.txt file shown in Fig.4 for implementation of Zeus and we got the infected host data in database. First of all the Botmaster is XAMPP and the other two victims machine with different operating system is for windows 7 and windows xp service pack 2 run infected bot.exe in their operating system. Both windows 7 and xp sp 2 is infected by Zeus. And the log of both infected machine is stored in botmaster database.

.

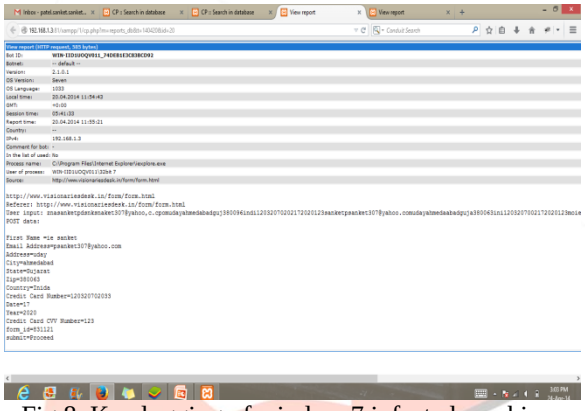

Fig.8. Key logging of window 7 infected machine.

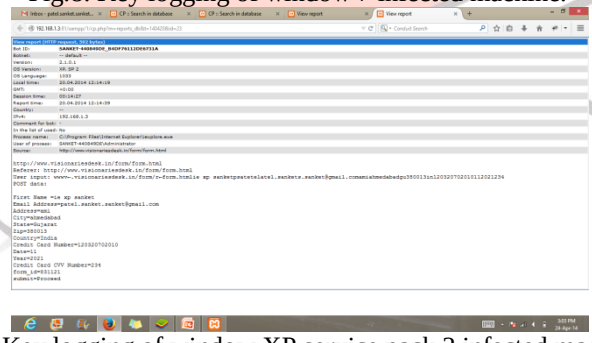

Fig.9. Key logging of window XP service pack 2 infected machines.

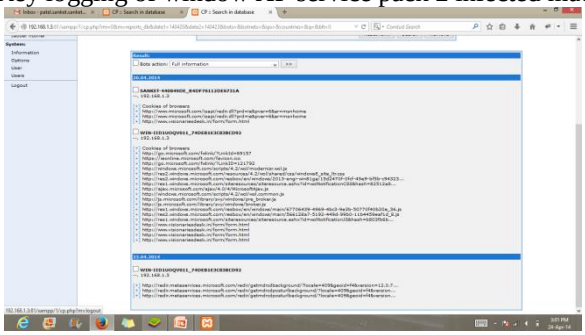

Fig.10. Botmaster database various infected machine data.

5. DETECTION OF ZEUS CRIME WARE USING WIRE SHARK

Data sent through the Zeus botnet is encrypted with RC4 encryption. In this implementation a key stream is generated from the botnet password, and is XORed with the data. The same password is used to encrypt all data that is passed through the botnet.

Changing the botnet password requires that all of the bot executables be updated to a build that includes the new password. The dynamic config file also must be updated and the server password changed from the Control Panel.

When a computer is infected with the Zeus bot, its first communication with the server is a request for the dynamic config file. Unlike other data sent through the network, the config file has already been encrypted by the Builder application and can be sent without further processing. Fig.11 shows the config file being requested by the bot, returned by the control server.

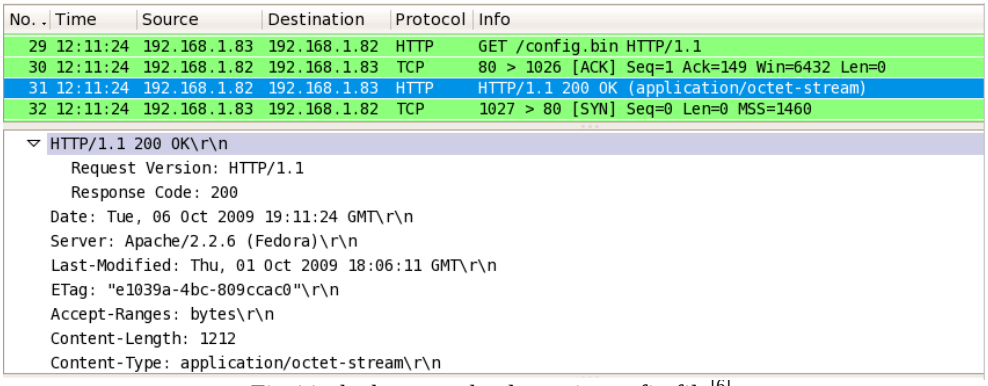

Fig.11. the bot gets the dynamic config file[6].

When the config file has been received, the bot will retrieve the drop server URL from it. The bot then HTTP POSTs some basic information about itself to the drop server, to log in and indicate that it is online. As long as it is running, the bot continues to HTTP POST encrypted logs and statistics to the server at timed intervals. By default logs are sent at 1 minute intervals and statistics are sent every 20 minutes.

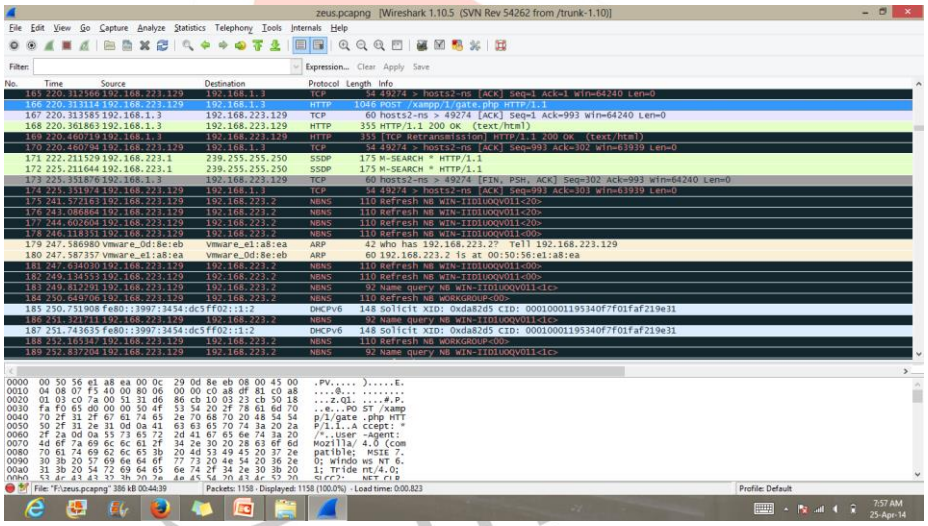

Fig.12. the Bot post basic information.

When a bot posts data to the server, the server replies with an HTTP/1.1 200 OK response. The Zeus server conceals an encrypted message as data within the response. This data field is used to send commands (scripts) to the bot. below is an example of the default data when no command is being sent, which is the most common case.

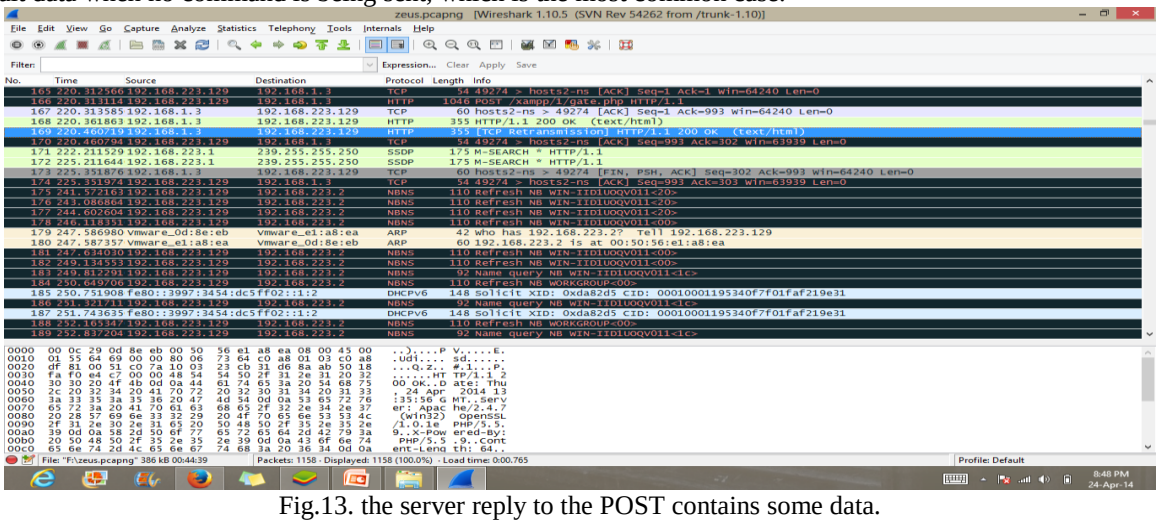

Fig.13. the server reply to the POST contains some data.

When a command script is being sent by the server, the data size will be considerably larger than the standard response. Fig.14 shows a sent getfile command, resulting in 64 bytes of encrypted data.

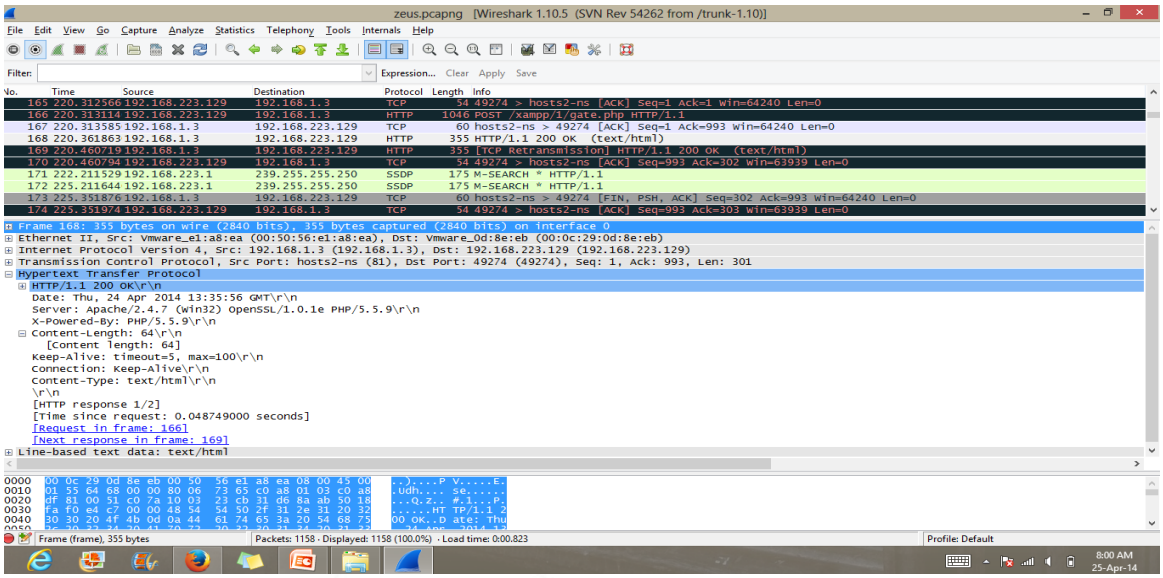

Fig.14. the server reply to the POST contains a command.

6. 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 in networking of the crime ware toolkit in general and about the malware binary in particular. In the stream of this thinking, we investigate malware binary file. To this end, we mainly employ "IDA Pro" [7] to disassemble the binaries and debug them to understand their business logic. The analysis is in one fold: First, the analysis that is linked to the malware binary file.

Zeus Bot Binary Analysis

As depicted in Fig.15, 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 Fig.16. 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. Fig.17 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 Fig.18.

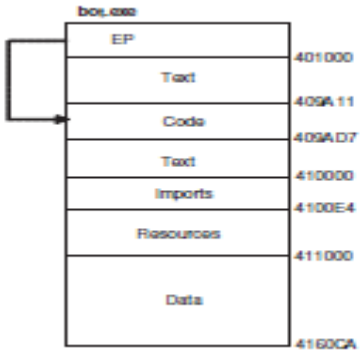

Fig.15. Segments of the Bot.exe binary file.

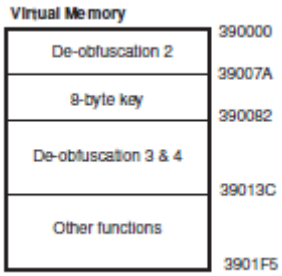

Fig.16.De-obfuscated code in the virtual memory.

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 Fig.19).
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

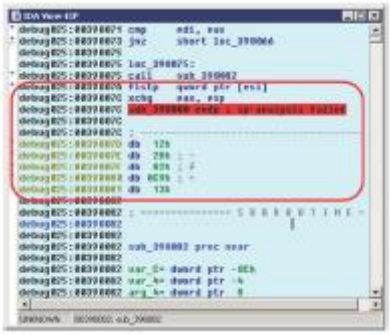

Fig.17.The 8-bit key.

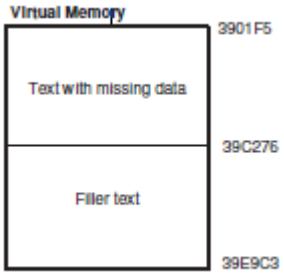

Fig.18.The virtual memory used by the second de-obfuscation routine.

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 1 and Algorithm 2[8].

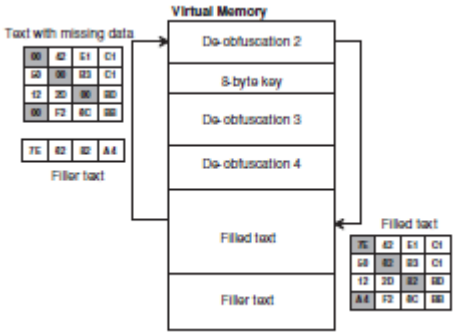

Fig.19.The result from the Second De-obfuscation routine.

Algorithm 1: DECRYPT_STRING(enc_string)
Seed=0xBA;
String new_string=new String(enc_string.length());
For i=0 to enc_string.length()
Then Do
new_string[i] = (enc_string[i]+seed)%256;
seed=(seed+2);

Return (new_string)

Algorithm 2: DECRYPT_URL(enc_url)
String new_url = new String(enc_url.length());
For i=0 to enc_url.length()
Then do
If(i%2 ==) 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)

2) Key Extraction[8]: 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 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 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 6-1. 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 [9], 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.

7. CONCLUSION

The Zeus crime ware 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.

REFERENCES

[1] Aniello castiglione, Robert De Priso, Alfredo De Santis, Ugo Flore, Francesco Palmieri, "A botnet based command and control approach relying on swarm intelligence", Journal of Network and Computer Application.

[2] Michael Bailey,Evan Cook,Farnam Jahanian,Yunjing Xu,Manish Karir "A Survey On Botnet Technology and Defensed", University of Michigan.

[3] Jivsh Govil,Jivika Govil,"Criminology of Botnets and their Detection and Defense Methods",IEEE EIT 2007.

[4] Craig A. Schiller, Jim Binkley, David Harley, Gadi Evron, Tony Bradley, Carsten Willems, Michael Cross," Botnets The Killer Web App".

[5] 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.

[6] http://bestbotnet.blogspot.in/2013/08/zeus-complete-tutorial-building-bot.html

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

[8] H. Binsalleeh, T. Ormerod, A. Boukhouta , P. Sinha, A. Youssef, M. Debbabi, L. Wang, "On The Analysis of the Zeus Botnet Crime ware Toolkit", National Cyber Forensics and Training Alliance Canada, Computer Security Laboratory, Concordia University Canada.

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