A Stateless Heuristic Approach to Detecting Encrypted Botnet Command & Control Traffic Over Standard HTTP POST

Fotios Lindiakos
flindiak@gmu.edu

Burns Mijanovich
bmijanov@gmu.edu

Greg Pothier
gpothier@gmu.edu

December 14, 2011

Abstract

In this paper we examine Hypertext Transfer Protocol (HTTP) POST data to see if it is feasible to differentiate legitimate traffic from botnet Command and Control (C&C) traffic without the need to decrypt said traffic. The premise behind this approach is that C&C traffic will most likely occur in the body of a POST message and be encrypted at the application layer to avoid detection. Through heuristic analysis we demonstrate that it is possible to differentiate unencrypted traffic from encrypted traffic in a stateless fashion. Furthermore, we show that encrypted traffic, while similar to encoded traffic (such as Multipurpose Internet Mail Extensions (MIME) file uploads) is differentiable from legitimate traffic due to its length and a similarly encrypted response. The result of this research is the development of a theoretical understanding of a new method for detecting botnet traffic. The outcome is the development of a detection engine capable of this with both minimal false positive and false negative rates.

1 Introduction

A botnet is a network of computers infected by a program that communicates with its' controller in order to perform unsolicited tasks. These groups of computers, infected with common malware, can be controlled by an outside entity. The term "bot" refers to the victim computers and the "master host" or "botmaster" refers to the controlling entity that sends instructions and receives information from the bots. After a computer is infected, the bot will send it's status to the command and control server and the bot will communicate with it to obtain updates and instructions. The earliest iterations of botnets typically used Internet Relay Chat (IRC) for command and control communication mostly because it scales well. However, as the threat and sophistication has evolved, many modern day botnets are gravitating towards HTTP communication because of its ubiquitous acceptance across networks and its support for encryption.

Botnets are now considered one of the greatest threats facing computer users today. Originally designed for benevolent purposes like automated administration tasks, bots have been

employed by malicious developers as nefarious tools used to steal banking information, log keystrokes, and exploit personal information. Botnets' sophistication, widespread distribution and dynamic nature pose a serious and real threat to Internet commerce and security. As this threat to commerce and end users continues to evolve, researchers are working to develop new ways to detect botnet malware and protect systems from exploitation.

Previous botnet research has focused on several areas including the passive monitoring of communication logs and traffic, analysis of ports and user characteristics, and even the development of algorithms to detect anomalies. Researchers have attempted to detect botnets by using flow data to detect idling clients to typical IRC ports as a methods of identifying potential bots [2]. The resource intensive nature of examining traffic flow over the network in real time has led some researchers to also sharpen their focus on analyzing secondary bot behavior as a means of detecting propogation and infection [3]. Other research has avoided the overhead of analyzing network traffic by focusing on the characterstics typical of specific botnets like using using unconventional ports for IRC communication or tracking bots based IRC nicknames. These methods are limited in modern day botnets as they have gravitated away from IRC communication and now utilize advanced techniques like fast flux, HTTP communication and encryption for data messaging. There has also been a focus on developing algorithms that can analyze large amounts of traffic summaries to detect botnet characteristics [4], however this has also been focused on IRC communication and does not tackle the encrypted HTTP messages seen in the real world today by such notorious botnets as Zeus.

2 Background of Zeus

We chose to analyze the Zeus botnet for several reasons. Zeus was labeled the king of botnets during 2009 and 2010, utilizing several attack vectors such as a trojans, banner grabbing, API hooking, and keystroke logging. In 2009, Zeus infected over 20 million computers and has been dubbed "the king of malware" [5].

The motivation behind the development of the Zeus crimeware kit is primarily financial as the botmaster can view credit card information and sensitive authentication data on all of the infected machines. In 2010, the Federal Bureau of Investigation (FBI) announced the discovery of a major international crime network using Zeus to hack into US computers to steal around $70 million. More than 90 suspected members of the ring were arrested in the US, and arrests were also made in UK and Ukraine. The Zeus botnet and crimeware kit is greater than just one criminal gang as there are several versions available for various prices.

The Zeus crimeware kit is comprised of a web application control panel, configuration files, a malware binary file, and a builder program. The control panel is the command center for the bot master where they communicate and issue orders to all of the botnet nodes. Zeus relies on this command and control communication to acquire sensitive information from the bots, provide updates to the bot, and push configuration changes from the botnet master server to the bots. This data is sent through HTTP POST messages utilizing RC4 encryption. The key stream is created using an Exclusive or (XOR) of the botnet password and the data that is sent. There is only one password responsible for encrypting all of the messages being sent throughout the botnet.

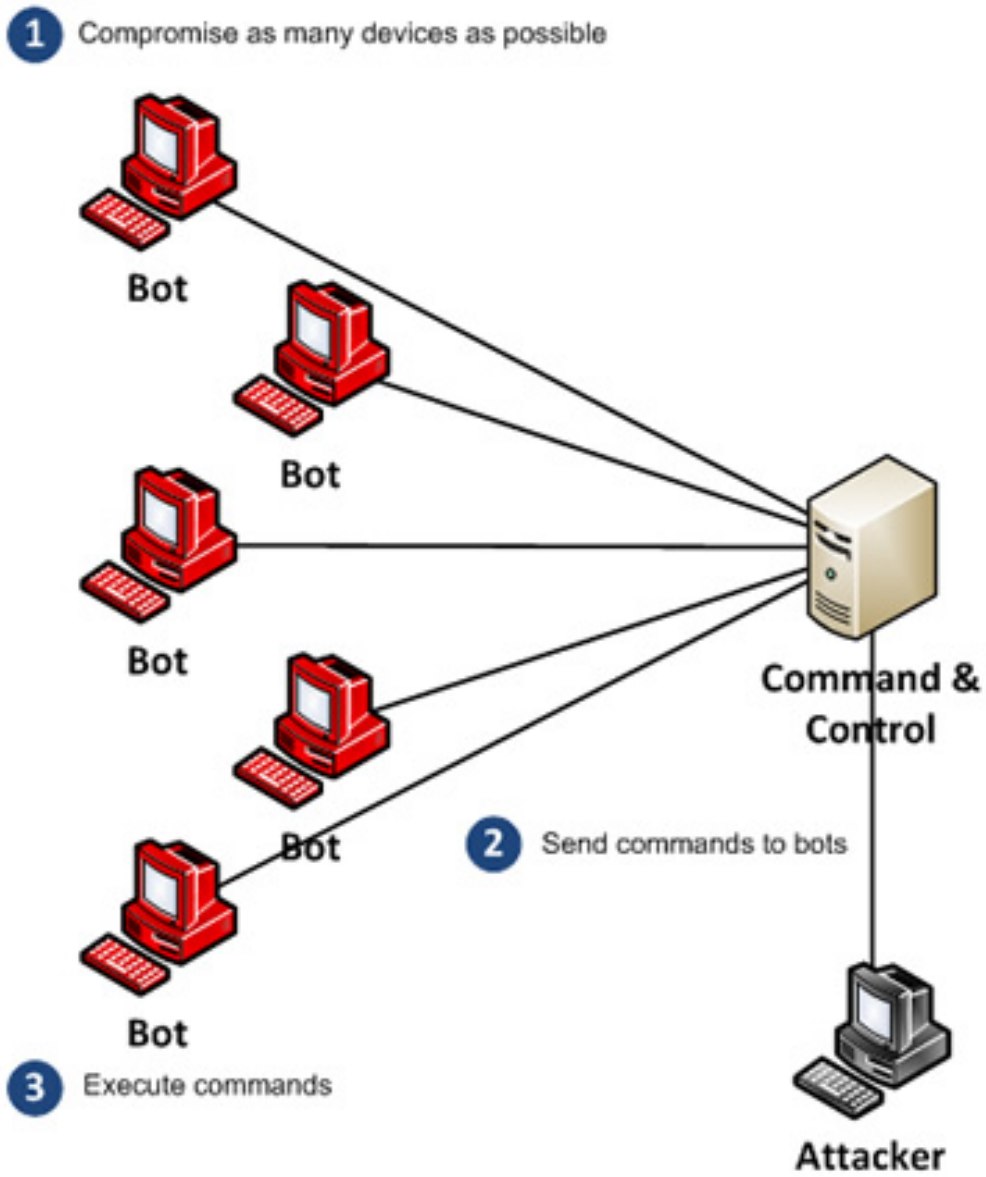

Figure 1: Typical botnet architecture [1]

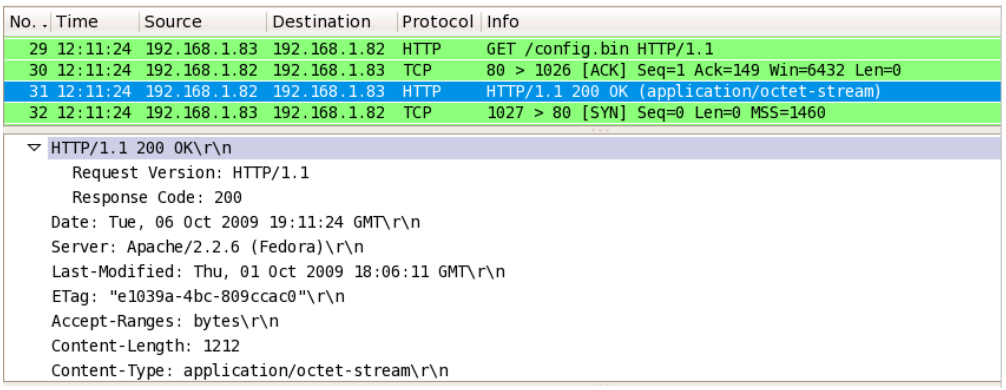

Figure 2: Packet capture of configuration file [6]

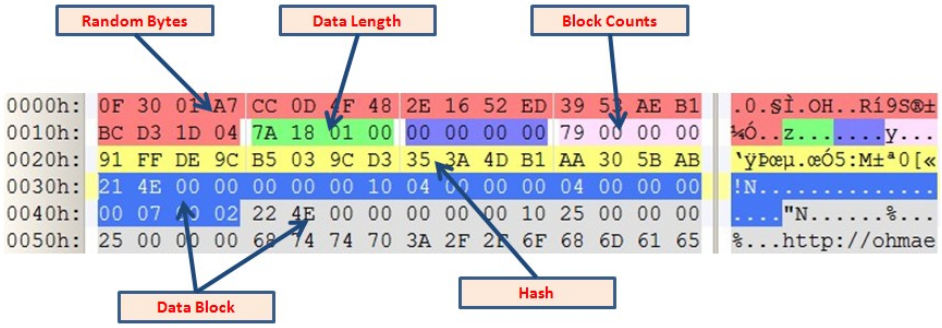

Figure 3: Zeus packet capture [7]

Upon initial infection, a new bot will first sent a message to the C&C server requesting a dynamic configuration file. The configuration file was already encrypted when the bot was initially built and the message sent can be seen in Figure 2.

Figure 3 is another screenshot of the configuration file being received by the bot. After the RC4 decryption of the message, the result isn't clear text but actually binary.

The server receives the request and sends the dynamic configuration file. This file contains the Universal Record Locator (URL) to the drop server. The bot responds by sending information about the compromised computer as well as an indication that it is now online. By default, the bot will send HTTP POST messages containing logs and information to the server every minute and statistics every 20 minutes. An example of this traffic can be seen in Figure 4.

Another capture of the bot sending the POST message containing information about the infected machine is seen in Figure 5.

The server receives the bot's POST message and replies with a `HTTP/1.1 200 OK` response. The message contains instructions (scripts) within the data which is encrypted. Figure 6 below displays a message being sent from the server without any encrypted instructions.

Figure 7 contrasts Figure 6 by illustrating a message that does contain instructions. The server messages containing instructions are larger than the standard server messages being

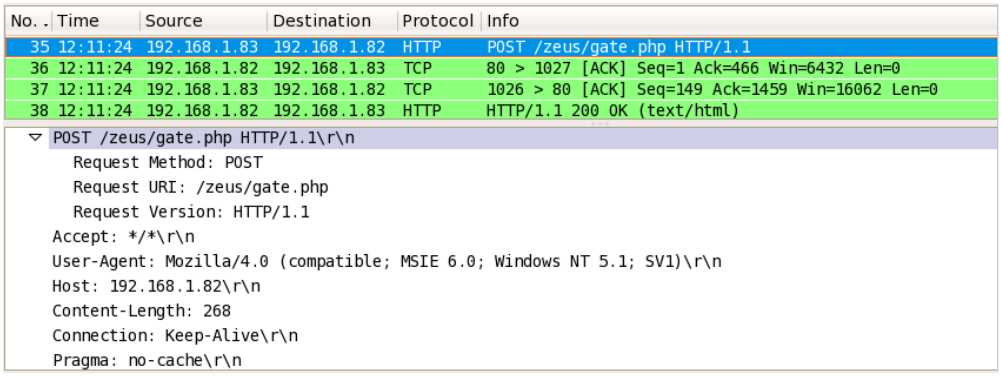

Figure 4: Zeus POSTs information [6]

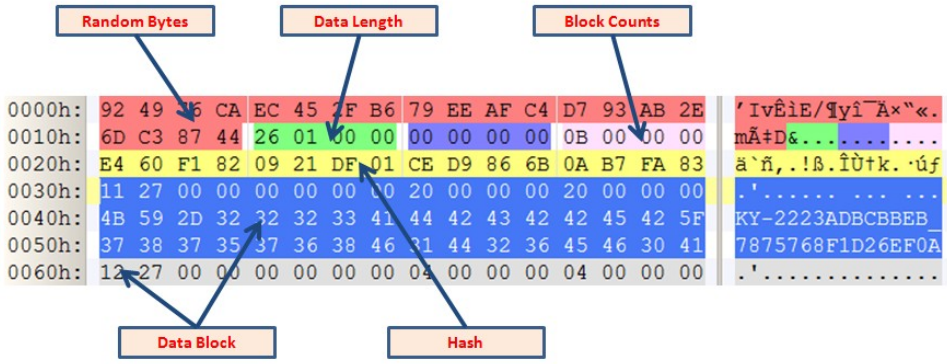

Figure 5: A detailed look at a Zeus packet capture [7]

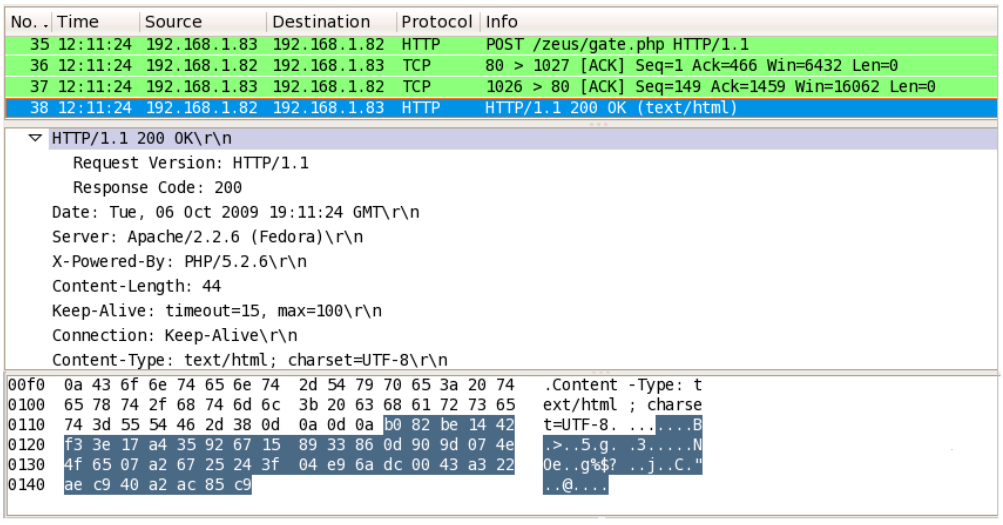

Figure 6: The server replies to the POST [6]

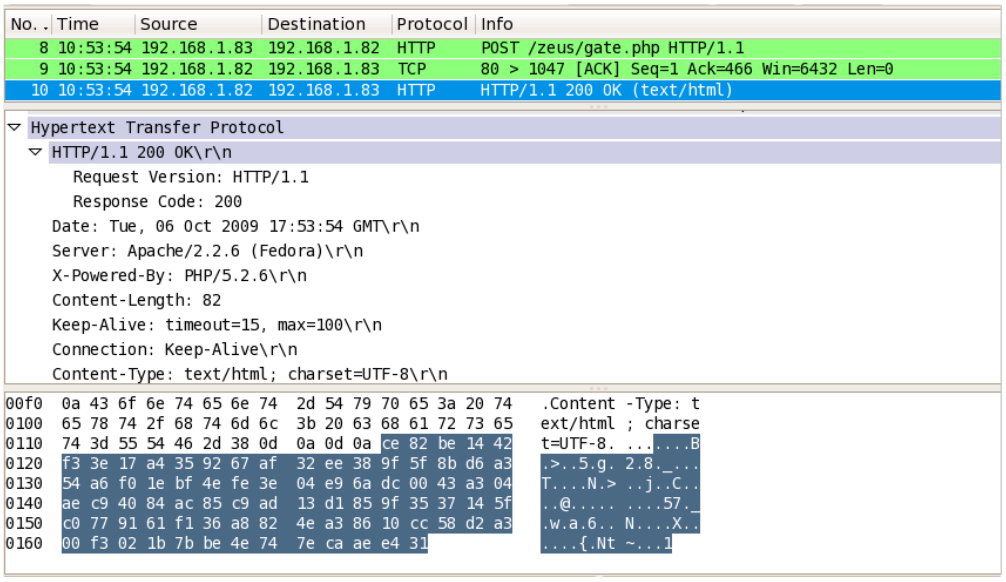

Figure 7: A server replies to the POST with commands [6]

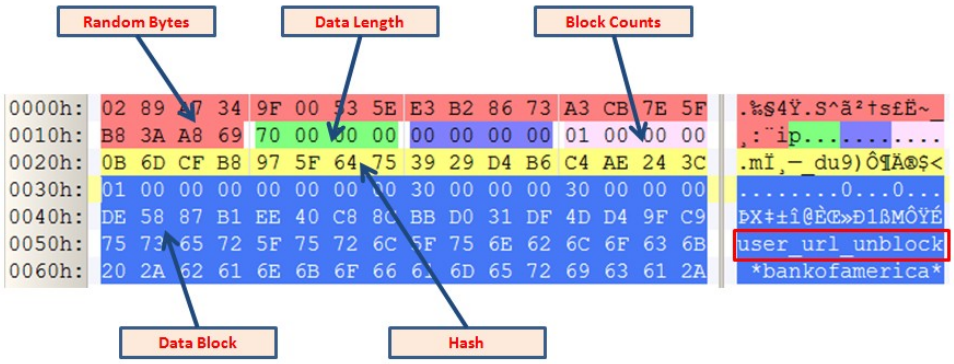

Figure 8: Another detailed look at a Zeus packet capture [7]

sent. The one in the figure below is bytes with a `Getfile` command.

Figure 8 shows another dissected packet of a message being sent from the server to the bot.

3 Statistical Analysis

The statistical methods we used to analyze the HTTP POST data included frequency analysis, index of coincidence calculation, and entropy calculation. These methods are frequently used in cryptanalysis to analyze encrypted data, especially in an effort to decrypt classical substitution ciphers. Although we are not attempting to decrypt the HTTP POST messages from the Zeus botnet, these methods also have value in distinguishing encrypted ciphertext from plaintext. We suspected that we could use some combination of these techniques to determine with high accuracy if a given HTTP POST was sourced from Zeus, or was normal

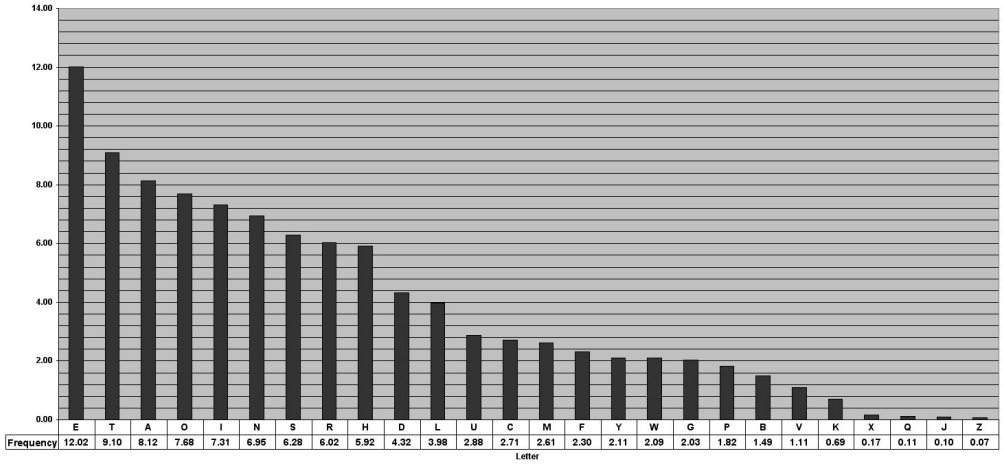

Figure 9: English Letter Frequency [8]

internet traffic. Because Zeus does not use Secure Sockets Layer (SSL), and encrypting just the POST data is highly unusual, we expected that, if successful, our approach would have a low false positive rate. What follows is an explanation of each of the statistical analysis methods we used.

3.1 Frequency Analysis

Frequency analysis is the most straightforward method we used to analyze the HTTP POST data. It involves counting the number of occurrences of each character or letter in a message. Historically, frequency analysis was used to crack classical monoalphabetic substitution ciphers by noting which letters appeared more frequently than others. This is possible because the frequency distribution of letters in English is not uniform: Some letters are far more common than others, as shown in Figure 9

Frequency analysis can also be performed on digrams (digraphs, or letter pairs) and trigrams (letter triplets); the English digram frequency distribution is shown in Figure 10 The principles of the analysis are the same, and in all cases it is theoretically possible to distinguish English text from random data, because the random data will have an even distribution. We posited that this would aid us in detecting HTTP POST messages by Zeus, because Zeus encrypts the POST data with RC4 before sending it over the network. RC4, like other modern symmetric cryptographic algorithms, produces data that appears random or near-random. Legitimate posts, while not conventional English text, generally contain at least some English words (names of form fields, etc.); perhaps enough to measureably shift the frequency distribution away from uniform randomness.

3.2 Index of Coincidence

The Index of Coincidence (IC) is a measure of how similar or dissimilar two blocks of text are. Classically, one would calculate the IC by counting the amount of identical characters in the same position in each text. IC analysis was used to break the polyalphabetic Vigenre cipher, a substitution cipher with multiple rotating alphabets depending on key size. One

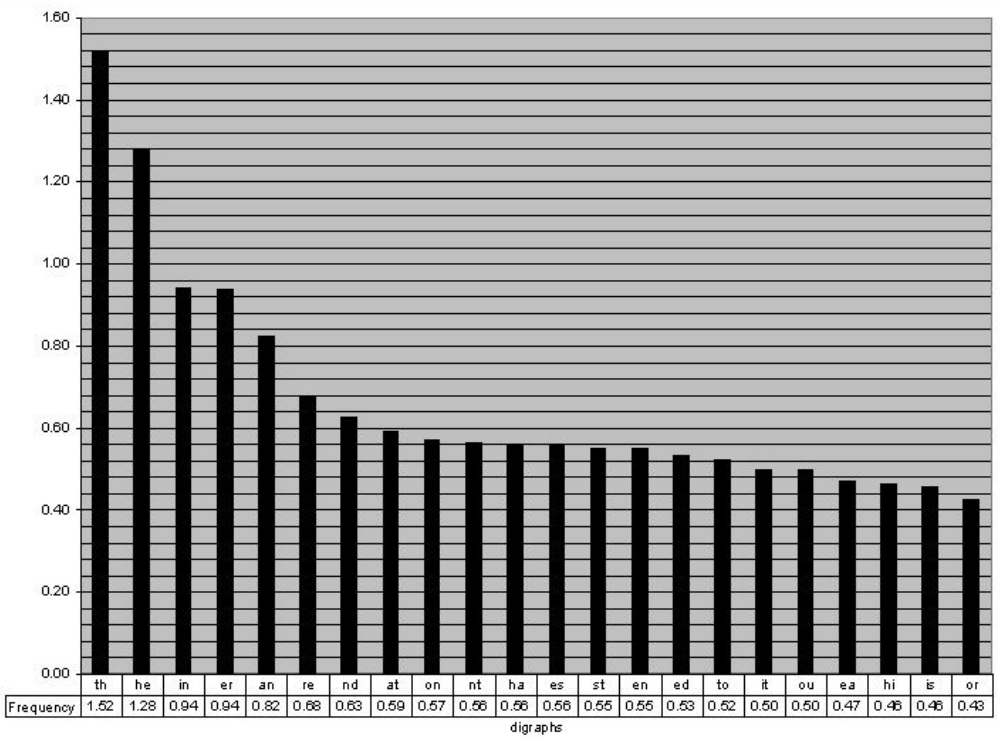

Figure 10: English Digram Frequency [9]

could determine the length of the key by calculating the IC of each chunk of text for each potential key length, and then the Vigenre cipher would be reduced to a series of simple Caesar (rotation) ciphers.

The IC is generally calculated as a probability of coincidence given a letter frequency distribution for the text block. This means that texts are not commonly compared character by character, rather that the IC for any given text block can be compared to the expected IC for English, derived from the letter frequency distribution for English. The IC can be calculated as show in Equation 1

$$IC = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)/c} \tag{1}$$

Where c is the number of letters in the language to be compared to (26 for English), n_i is the integer frequency of a given letter, and N is the sum of all n_i. The results of this equation can be compared against the expected IC for a given language, in our case English, found by using Equation 2.

$$IC_{expected} = \frac{\sum_{i=1}^{c} {f_i}^2}{1/c} \tag{2}$$

Where c is the number of letters in the language in question (26 for English), and f_i is the relative frequency of a given letter. By this equation, the IC for English is 0.067 (after normalizing by dividing by c).

We suspected that malicious HTTP POST data from Zeus could be distinguished from legitimate POST data by comparing the IC for both sets of data. The IC for the legitimate data will likely not match English exactly, because the messages will likely be short. However, it may be a much closer match than the Zeus POSTs, because Zeus uses encryption, so the frequency distribution for those POSTs should be much more uniform. We expected the Zeus POST messages to have an IC much closer to 0.0385, or $\frac{1}{26}$, a totally uniform distribution.

3.3 Entropy

Entropy is a measurement of unpredictability in data. Specifically, entropy measures the uncertainty of how one unit of information, such as a bit or a byte, might effect subsequent units. Thus, data achieves maximum entropy when each bit is totally independent from the others and totally random. In this case, each bit has one bit of entropy. This concept of entropy was developed by the mathematician Claude Shannon, and the Shannon entropy can be calculated as shown in Equation 3.

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i) \tag{3}$$

Where H is the entropy, X is the message, n is the number of possible values, b is the base of the logarithm (2 in our case, since we are dealing with binary data), and p is a function called the probability mass function. The probability mass function determines the probability of the next character individually, irrespective of the previous character; this is a function of the frequency distribution of the message and easily calculated.

Because encrypted data has high entropy, and the entropy of unencrypted data is comparatively lower, we believed that we could use entropy to differentiate between malicious POST data from Zeus, and legitimate traffic. In addition, we noticed that the encrypted Zeus POSTs had a high number of non-printing characters: This is because encryption algorithms such as RC4 do not attempt to make their ciphertext output conform to the ASCII printable character set. Because of this, we also calculated the percentage of printable ASCII characters in each HTTP POST to attempt to flag malicious Zeus traffic.

4 Implementation

While trying to develop and implement a solution, our first order of business was to gather a corpus of network traffic, both "good" and "bad" (which in our case meant containing Zeus C&C traffic). We were able to find publicly available Packet Capture (PCAP) files of representative network traffic, files of Zeus traffic, and one of the team members captured his network traffic for a day (with SSL stripped). [10, 11]

Then, in order to calculate the aforementioned statistical values for POST packets, we implemented the tests in Ruby and then wrote a program to iterate over all of the PCAP files, rendering the results into a Comman-separated values (CSV) file. We then wrote another Ruby program that would calculate the number of packets that would be rejected given certain thresholds for our statistical values.

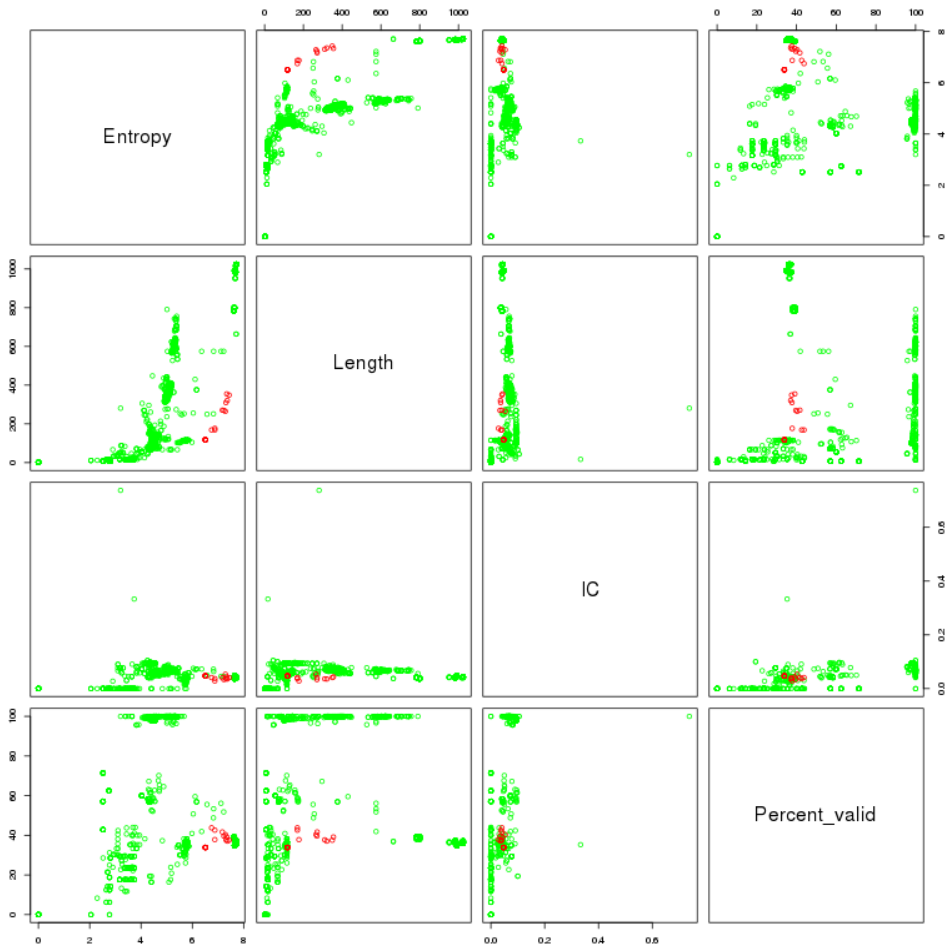

Figure 11: Pairwise Scatterplot of POST data statistics

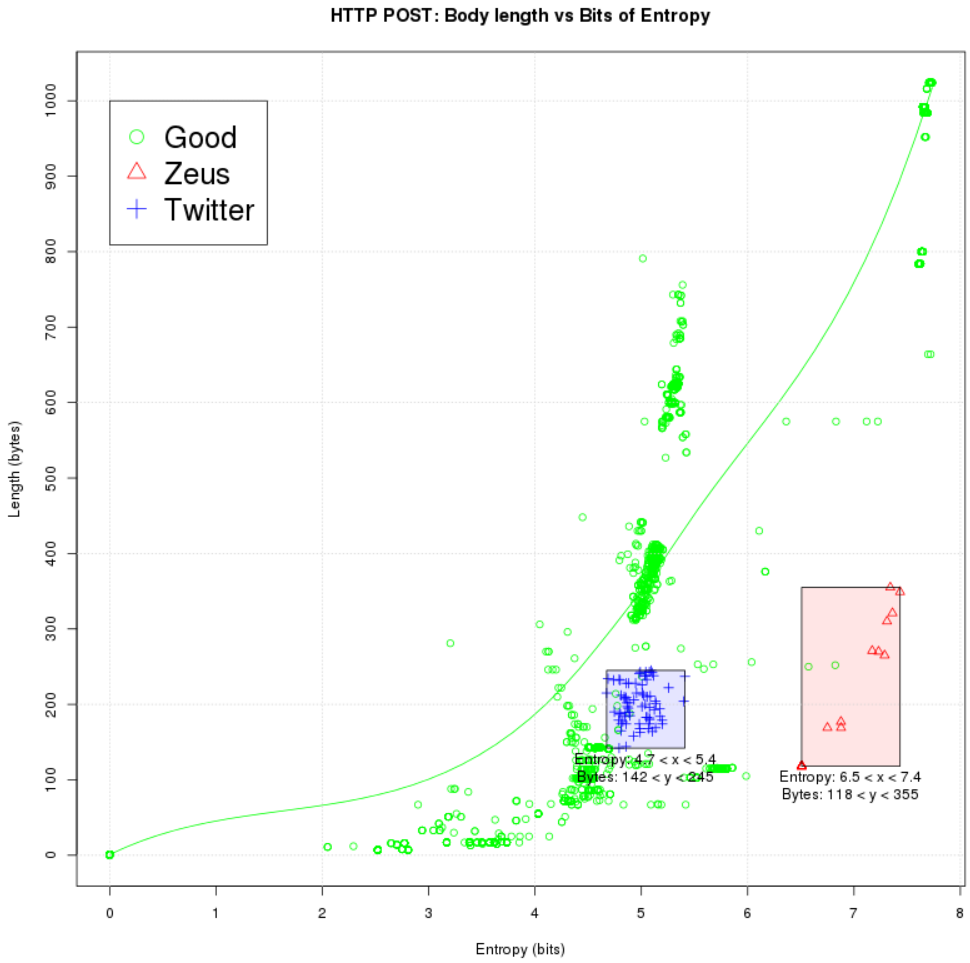

Figure 12: HTTP POST: Body Length vs Bits of Entropy

When we finished calculating all of the statistical values for the POST data, we generated graphs to visually compare the multiple factors, as seen in Figure 11 The green dots represent "Good" traffic, while the red dots represent Zeus-bot traffic. By comparing two factors, we determined that "Length vs Entropy" gave us the most clear distinction between traffic with an adequate buffer between "good" and Zeus traffic.

Upon further analysis of the length and entropy of the POST data, it became very clear that there was a clear distinction between Zeus traffic and other traffic. For another dataset, we also included sample HTTP POST messages to the popular micro-blogging site, Twitter. Other traffic that we analyzed, but did not fit on Figure 12 was image uploads to the popular image sharing site, Flickr. These files were all well over 100,000 bytes. The value was the number of bytes in the image, plus an additional 1600 bytes of overhead data. The values were also very tightly grouped with an entropy very near 8. The grouping in Figure 12 further demonstrates that Length vs Entropy is a very good indicator of network traffic for a particular site or application.

Based on Figure 12, we set thresholds for Length and Entropy, and calculated the number

Type	Packets	Length	Entropy	Combined	FPR	FNR
Good	9199	1277	5538	2	.0002	
Zeus	20	20	20	20		0

Table 1: Packets filtered based on specified thresholds

of packets that would be blocked based on the criteria specified, as seen in Table 1. If length or entropy were taken individually, the False Positive Rate (FPR) would be unacceptably high, but when they are combined, just 2 out of over 9000 packets would be falsely flagged. Also, based on our dataset, these thresholds would detect all Zeus packets.

5 Conclusion

By combining two disparate statistics about an HTTP POST packet, we have determined that it is feasible to segregate Zeus (and potentially other malware) traffic from legitimate traffic. Since length is already a standard Transmission Control Protocol (TCP) header field, using it as a first, fast filter would cut down the workload significantly. In our test data, the number of packets was reduced by about 85 percent. Then creating a filter for calculating Shannon Entropy should be trivial as the formula has been around for over 50 years, and undoubtedly implemented in all popular programming languages. The only downside to this approach is that in order to calculate the entropy of the packet, the Intrusion Detection System (IDS) will need to have access to the entire TCP stream, which may require reconstructing it, depending on Maximum Transmission Unit (MTU) and segment size. Fortunately, if length is taken into account first, only relatively small sized packets will need to be reconstructed. We believe that this experiment demonstrated that statistical analysis of network traffic can be helpful in detecting botnet and malware activity, and we hope that this approach can be implemented and used in the future to detect other types of malware.

References

[1] S. Gandhi, "Junk ssl connection enables to target major websites by botnets," Dec. 2011. [Online]. Available: http://bligbook.com/junk-ssl-connection-enables-target-major-websites-botnets/

[2] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in dns traffic," in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, oct. 2007, pp. 715 –720.

[3] E. Cooke, F. Jahanian, and D. Mcpherson, "The zombie roundup: Understanding, detecting, and disrupting botnets," in *1st Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2005, pp. 39–44.

[4] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07)*, 2007.

[5] R. Garvey, "Zeus (still) wants your wallet," *eSecurity Planet: Internet Security for IT pros*, Nov. 2001. [Online]. Available: http://www.esecurityplanet.com/hackers/zeus-still-wants-your-wallet-.html

[6] D. MacDonald, "Zeus: God of DIY botnets," *FORTINET: Threat Research and Response*, 2011. [Online]. Available: http://www.fortiguard.com/analysis/zeusanalysis.html

[7] K. Yang, "Zeus: Bot to master early communication protocol (part two of two)," *FORTINET: Threat Research and Response*, Nov. 2011. [Online]. Available: http://blog.fortinet.com/zeus-bot-to-master-early-communication-protocol-part-two-of-two/

[8] L. Smithline and T. Wall, "English letter frequency (based on a sample of 40,000 words)," 2004. [Online]. Available: http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html

[9] ——, "Digraph frequency (based on a sample of 40,000 words)," 2004. [Online]. Available: http://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/digraphs.html

[10] "Publicly available PCAP files." [Online]. Available: http://sourceforge.net/apps/mediawiki/networkminer/index.php?title=Publicly_available_PCAP_files

[11] "Openpacket.org: Capture repo - malicious." [Online]. Available: https://www.openpacket.org/capture/by_category?category=Malicious