Detecting Encrypted Botnet Traffic

Han Zhang, Christos Papadopoulos, Dan Massey
Computer Science Department,
Colorado State University,
Fort Collins,
United States,
zhang@cs.colostate.edu, christos@cs.colostate.edu, massey@cs.colostate.edu

***Abstract*—Bot detection methods that rely on deep packet inspection (DPI) can be foiled by encryption. Encryption, however, increases entropy. This paper investigates whether adding high-entropy detectors to an existing bot detection tool that uses DPI can restore some of the bot visibility. We present two high-entropy classifiers, and use one of them to enhance BotHunter. Our results show that while BotHunter misses about 50% of the bots when they employ encryption, our high-entropy classifier restores most of its ability to detect bots, even when they use encryption.**

I. Introduction

Recently, some botnets have started using encryption to evade detection. Examples include Storm and Nugache, two very popular P2P botnets. Every message in Storm is XOR encrypted [7],[10]. Nugache uses a variable bit length RSA key exchange, which is used to seed symmetric Rijndael-256 session keys for subsequent peer communication [12]. Moreover, all the Command & Control (C&C) traffic in Rustock is encrypted using RC4 [3].

In [15] the authors design an advanced hybrid peer-to-peer botnet with several new features to make the botnet hard to detect and shut down. One of the proposed features is to let each bot randomly generate a symmetric encryption key for communication. In [14] the authors evaluate the evadability of botnet detection systems, and present several approaches a botnet can use to defeat detection systems. One of the approaches encrypts connections between the botmaster and bots and among the bots themselves.

In this work we explore the use of entropy as a feature to detect encrypted bot communication. We show that encrypted traffic increases the entropy of a flow and we build detectors for high-entropy flows. Many types of traffic, however, have high entropy, including media, executables and compressed files. Therefore, high entropy alone does not imply encryption. To avoid false positives, we add our entropy detectors to an existing bot detection tool, namely BotHunter[6], which implements an event-based bot detection engine. Many of BotHunter's events, however, are triggered using deep packet inspection (DPI), which unfortunately becomes blind with encryption. We demonstrate that by enhancing BotHunter with our entropy detectors we can restore virtually all of BotHunter's ability to detect bots. Our contributions are as follows:

1) We show that bot detection systems such as BotHunter that rely on DPI suffer significantly in the presence of encrypted traffic, its detection rate reduced by almost 50%.
2) We show that encryption produces high entropy flows and build two detectors to identify such flows.
3) We extend BotHunter with our detectors and restore its ability to detect bots.

The rest of paper is organized as follows: Section II describes the datasets used in our experiments. Section III discusses BotHunter and its scoring system. In Section IV, we apply encryption and IP anonymization on botnet traffic and analyze how they affect the detection performance of BotHunter. We investigate entropy of encrypted traffic and introduce two approaches to detect high-entropy connections in Section V. Finally we enhance BotHunter with our detectors in Section VI and evaluate true and false positives. We discuss related work in Section VII before we conclude in Section VIII.

II. Datasets

We used two groups of datasets in our experiments. The first group includes 140 Butterfly and Kraken botnet traces obtained from Georgia Tech[4], [11]. The second group includes three traces captured at our university lab. All traces are in tcpdump format and include payload. For convenience, the botnet dataset and lab datasets are named *140Samples*, *Lab1*, *Lab2* and *Lab3*.

Each trace in *140Samples* is collected by running captured bot binaries in an isolated environment. These traces have roughly similar communication patterns, namely send GET requests to download several executable files, and then GET requests to C&C servers to download C&C data. Finally, the bots start sending spam.

Lab1, *Lab2* and *Lab3* were captured in our lab, which has over 100 users and is connected to the outside world via a 10Gb/s link. Our lab has a dedicated /24 subnet and runs standard services, such as mail, web, ssh, etc., and also carries traffic for several research projects including two planetlab nodes and a GENI rack. We used traces from out lab because this is the only place we can collect traces that include payload. The first two traces are 24-hour traces captured on Feb-22th-2011 and Dec-11th-2012, respectively. *Lab3* is a 72 hour trace captured on Dec-13th to Dec-16th-2012. Some statistics for these datasets are given in Table I. Besides, the peak bandwidth for *Lab1*, *Lab2*, *Lab3* is 29.3Mb/s, 133.9Mb/s and 81.9Mb/s.

III. BotHunter Background

BotHunter[6] (BH) is real-time bot detection system. Briefly, the fundamental observation in BH is that bot infection

follows a distinct set of events in some loose order. In its original form, BH included five infection events, such as *Inbound Scan*, *Inbound Infection*, *Egg Download*, *C&C communication* and *Outbound Scan*. In its latest incarnation (Version 1.6), BH includes eight events, listed in Table II [1]. BH builds a detection system that scores hosts engaging in such activity and flags them as bots if the score exceeds a threshold (0.8 in the current implementation).

BH makes use of snort. It introduces specific snort rules where appropriate, to detect events of interest and then uses the resulting snort output to aid in bot detection. Some of the snort rules apply DPI to search for specific patterns, such as magic numbers to detect executable downloads or signature strings associated with known bots.

A. BotHunter Scoring System

As mentioned earlier, each event is associated with a score. Since the source code for BH was not available to us, it is hard to determine these scores directly. BH generates no host-specific output if the final score is less than the threshold, and even when a host profile is generated, only the total score is given. To determine the individual event weights, we apply reverse engineering techniques as follows: we disable the snort rules that correspond to specific events, and then measure the difference in the final score between host profiles associated with the disabled event. For example, if the original score without disabling any snort rules is 3.5 and the score after disabling the rules to detect *Egg Download* is 3.0, then we conclude that *Egg Download* has a weight of 0.5. Our bot traces trigger six out of eight events in BH and the weights are shown in Table III.

TABLE I
LAB TRACES

	Lab1	Lab2	Lab3
TCP	93.55%	51.87%	72.25%
UDP	1.03%	2.83%	0.258%
ICMP	5.41%	45.14%	27.27%
Others	0.001 %	0.149%	0.215%
Bandwidth	14.08 Mb/s	26.16 Mb/s	18.4Mb/s
Duration	24 hours	24 hours	72 hours

TABLE II
EVENTS IN LATEST BOTHUNTER

Dialog Class	Dialog Event	Description
E1[bh] E1[rb]	Inbound Scan	External to internal inbound scan based on behaviors and rules
E2[rb]	Inbound Attack	Host is the target of inbound infection attempt
E2[dns]	DNS Lookup to Client Exploit	Host performs DNS query to malicious host associated with client-side exploits
E3[rb]	Egg Download	Host downloads binary executable from external host
E4[rb]	Malware C&C	Host exchanges C&C messages with remote controller
E4[nbr]	Russian Business Net Connection	Host connects to monitored Russian Business Network site
E4[dns]	DNS Lookup To Botnet C&C	Host performs DNS query to known malware control site
E5[rb]	Outbound Attack Propagation	Host conducts outbound attacks to propagate malware infection
E5[bh]	Outbound Scan	Host performs outbound scanning
E6[rb]	Attack Preparation	Host performs activities to launch attack
E7[rb]	P2P Coordination	Host has P2P communications associated with Malware coordination
E8[bh]	Malicious Outbound Scan	Host performs scanning using network ports associated with malware propagation
E8[rb]	Outbound to Malware site	Host connects to known malware site

TABLE III
BOTHUNTER SCORES

Event Alert	Score
Egg Download	0.5
C&C Communication based on DPI	0.5
C&C Communication based on RBN	0.4
Outbound Scan based on DPI	0.5
Outbound Scan based on Behavior	0.3
Attack Preparation	0.5
Bot Declaration	0.8

IV. BOTHUNTER AND ENCRYPTED TRAFFIC

As mentioned earlier, BH utilizes eight events to make a decision. Examining the rules for these events we determine that most of them rely on DPI and blacklists of IP addresses. This means that a botnet may be able to evade detection by encrypting or scrambling content and changing the IP addresses it uses. In this section, we analyze how these evasion techniques affect BotHunter.

To help in our evaluation, we implemented a *libtrace* tool that reads a network trace (in tcpdump format), encrypts the payload and anonymizes IP addresses. Our goal is to eventually provide many encryption options and allow users to specify which they want. However, in the current implementation we only provide simple XOR encryption, but allow users to specify the key. XOR encryption has a very low barrier of entry. With XOR, the tool treats each packet separately, in other words, it reads a packet, applies XOR to the payload, applies any requested IP address changes and writes the packet into a new file. This operation does not alter the packet size, although it does change the checksum. We plan to enhance our tool to re-calculate the checksum. For now, we simply instruct snort to ignore it.

We use the dataset *140Samples* and the latest version of BotHunter to perform four experiments with different combinations of payload encryption and IP anonymization. The results are shown in Table IV.

TABLE IV
BOTHUNTER DETECTION RATE

Traffic Encryption	IP Anonymization	Detection Rate
No	No	140/140
No	Yes	139/140
Yes	No	77/140
Yes	Yes	0/140

From Table IV we can see that BH detects all the bots in the original traces. BH also detects almost all the bots after applying IP anonymization (neutralizing the black list), missing only one. However, only 77 out of 140 bots are detected after applying payload encryption (but no IP anonymization). Detection rate drops to zero when we apply both payload encryption and IP anonymization.

To understand these results we divide the 140 samples into two groups. The first includes the 77 samples which are detected even after payload encryption, and the second contains the remaining 63. We answer the following three questions: i) Why is group 1 detected when the payload is encrypted, ii) Why group 2 is not detected when the payload is encrypted, and iii) Why is group 1 not detected when applying both payload encryption and IP anonymization.

All the original samples (with no payload encryption or IP anonymization) in group 1 are flagged as bots in very similar ways. More precisely, the following events are triggered: *Egg Download*, *C&C Traffic*, *Outbound Scan*, *C&C Traffic (RBN)*, *Outbound Scan (spp)* and *Bot Declaration*. Within these events, the former three rely on DPI techniques while the latter three rely on IP blacklist or behavior, which are not affected by payload encryption. The total score of the latter three events is 1.2, which is still greater than the detection threshold (0.8). As a result, these samples are still detected as bots even when payload is encrypted.

The group 2 bots (no payload encryption but with IP anonymization) are detected in similar ways with the following events: *Egg Download*, *C&C Traffic*, *Outbound Scan*, *C&C Traffic (RBN)* and *Outbound Scan (spp)*. The former three events are detected using DPI techniques so they are blind when payload is encrypted. The latter two don't rely on DPI techniques so the events are still triggered when payload is encrypted. However, the total score of these two events is only 0.7, which is less than 0.8, so the hosts are not identified as bots.

With both payload encryption and IP anonymization none of the 140 samples is detected as a bot. The reason is that payload encryption fails events *Egg Download*, *C&C Traffic* and *Outbound Scan*. IP anonymization fails events *C&C Traffic (RBN)* and *Bot Declaration*. Neither payload encryption nor IP anonymization affects detection of event *Outbound Scan (spp)*, which is triggered, but its score is only 0.3 and thus below the detection threshold.

V. High Entropy Flow Detection

In the last section we showed that encrypted payload can evade detection systems such as BotHunter that rely on DPI. Encryption, however, tends to alter the payload entropy, often converting low entropy (LE) payload to high entropy (HE). Recall that our premise in this work is that the presence of at least one HE flow along with other features that BotHunter detects is a reliable detector of encrypted malicious traffic. But how does one determine if a flow is HE?

In this section, we attempt to answer this question by introducing two algorithms to detect HE flows. The first calculates entropy across all user data in a flow and declares the flow HE if entropy exceeds a threshold. The second, calculates entropy of each individual packet, marks each packet as HE if its entropy exceeds a threshold, and the whole flow as HE if a sufficient percentage of packets are marked HE. While the flow-based algorithm seems more direct, there are some important reasons to consider the packet-based algorithm: (a) from an implementation standpoint it is easier to implement the packet-based algorithm since the entropy calculator does not need to maintain per-flow state; and (b) a packet-based algorithm is more tunable and potentially resilient to the occasional LE packet, as found for example, in protocol or file headers.

A. Entropy Background

In information theory, the definition of entropy was first introduced by C.E.Shannon [13] and has been used widely in various areas. Here we present a very quick introduction to entropy, followed by a discussion on how to decide whether a packet has high or low entropy by using the approach described in [9].

Let *w* be a word composed of N characters over alphabet $\Sigma = (a_0, a_1 ... a_{m-1})$. For each $a_i \in \Sigma$, we can count its occurrences, denoted as n_i, and then the frequency f_i of a_i is calculated as n_i/N. The entropy of *w* is estimated as:

$$\hat{H}_N^{MLE}(w) = -\sum_{i=0}^{\infty} f_i log f_i$$

where *MLE* is the maximum likelihood estimator. When the characters are bytes, the maximum entropy is 8 (bits per byte). The above estimator is a little different from the original definition of entropy, which is:

$$H(p) = -\sum_{i=0}^{m-1} p_i log p_i$$

However, the approximation $\hat{H}_N^{MLE}(w) \sim H(p)$ is valid when $N >> m$. In practice, the condition $N >> m$ could be an issue because some packets are short and less than m. [9] introduces an approach to solve this problem. Firstly, the authors estimate the average sample entropy $H_N(p)$, which is defined as average of $\hat{H}_N^{MLE}(w)$ over all words *w* of length N. When *p* is uniform distribution μ, $H_N(\mu)$ can be calculated as:

$$H_N(\mu) = log_2^m + log_2^c - e^{-c} \sum_{j=1}^{+\infty} \frac{c^{j-1}}{(j-1)!} log_2^j + o(1)$$

with $c = \frac{N}{m}$ and $o(1)$ is less than 0.004.

Similar to [9], we use the Monte-Carlo method to estimate the standard deviation $SD(\hat{H}_N^{MLE})$. Finally, we use $H_N(\mu) - 4 * SD(\hat{H}_N^{MLE})$ as the entropy threshold to decide whether the data under investigation is high or low entropy.

Note that the above methodology applies to both the entire flow and individual packets. In the former case we consider data in the entire flow, while in the latter we consider the payload of each packet individually.

B. HE Flow Detection

We now present the flow- and packet-based algorithms to designate a flow as HE. We will then compare these algorithms by applying them on data of known, popular types.

While they encrypt user data, many encryption protocols exchange packets at the beginning of a flow that are not encrypted (and thus low entropy). After the initial exchange,

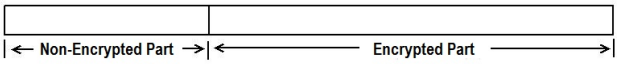

Fig. 1. Protocol Format 1

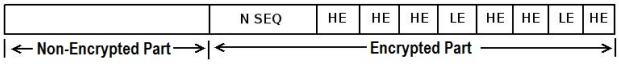

Fig. 2. Protocol Format 2

subsequent packets are encrypted. Figure 1 illustrates this scenario. To prevent such LE packets from skewing entropy calculation our algorithms wait until *N Sequential High Entropy Packets* have been detected before calculating entropy. Unfortunately, there is no clear way to estimate N, so we determine the value of N experimentally. For our datasets N = 2 seems to work best.

We now describe briefly the flow-based and packet-based algorithms. Recall that both algorithms aim at labeling a flow as HE or LE, but the former does so by examining the entire flow data, where the latter examines each packet separately.

1) Flow-based Entropy:
   This algorithm is very simple: we begin by calculating the entropy of each packet (as described earlier). After detecting *N Sequential High Entropy Packets* we capture the payload of all subsequent packets and then calculate the cumulative entropy of the resulting data (including the initial HE packets). We then compare the cumulative entropy with the threshold, as described earlier. If the cumulative entropy is greater than the threshold, then the flow is identified as HE, else it is LE. [1]
2) Packet-based Entropy:
   After detecting *N Sequential high entropy packets*, we calculate the entropy for each packet and classify it as HE or LE. At the end of the flow we count the number of HE and LE packets, denoted as *N(HE)* and *N(LE)*. If *N(HE)/(N(HE)+N(LE))* is greater than our threshold, which is named *High Entropy Packet Percentage Threshold*, then we consider the flow as HE. Figure 2 illustrates this approach.

C. Comparing Flow and Packet-based Entropy Classification

In this section we carry out sanity checks for our entropy classification algorithms. In the first test, we encrypt several popular types of files and run both entropy algorithms to validate our assumption that encrypted data is typically HE. In the second test, we run our algorithms on network flows we expect to carry encrypted traffic, specifically SSH and HTTPS flows, and expect most of them to be classified as HE. Finally, in the third test we run the algorithms on known non-encrypted (but not necessarily LE) traffic to ensure we don't see any anomalous results.

[1]While we could generalize the algorithm to detect N initial HE bytes, in reality most protocol interaction is done in units of packets, so we felt that our approach is appropriate.

1) Test 1: Encrypted Files: We carry out this test using 150 files each of MSword (.doc), MS Excel (.xls), ASCII (.txt) and PDF files for a total of 600 files. We encrypt all 600 files using 17 commonly used encryption algorithms. To simulate network traffic, the encrypted files are divided into 1500 byte chunks. We use N = 2 as *Sequential High Entropy Packets* and 90% as the *Entropy Packet Percentage Threshold*.

Table V shows the results. We can see that both flow and packet-based algorithms perform very well in classifying encrypted data as HE, with the packet-based algorithm offering slightly more hits. This is an encouraging result, indicating that encrypting a file does indeed result in HE. The table also shows that only 6 and 32 out of 600 files are classified as HE before encryption, meaning that encryption does, in general, alter entropy.

TABLE V
OFFLINE ENCRYPTED FILES

Encryption Algorithm	Key Length (bit)	HE (flow-based) out of 600	HE (packet-based) out of 600
None	N/A	6	32
DESX	128	599	599
Blowfish	448	600	600
Rijndael (AES)	256	600	600
CAST	256	600	600
Triple-DES	192	600	600
RC2	1024	599	600
Diamond 2	2048	598	600
Tea	128	599	600
Safer	128	599	600
3-Way	96	600	600
GOST	256	600	600
Shark	128	599	600
Square	128	598	600
Skipjack	80	598	600
Twofish	256	598	600
MARS (IBM)	448	600	600
Serpent	256	600	600

2) Test 2: Encrypted Traffic: In this test we repeat the above methodology, but this time using real network traffic, specifically the trace *Lab1*. We isolate traffic on ports 22 (SSH) and 443 (HTTPS). The results are shown in Table VI. As we can see from the table the two algorithms identify 95% and 97% of the flows in the trace as HE, a very good result.

TABLE VI
ONLINE ENCRYPTED TRAFFIC

Traffic	SSH	HTTPS
Total Flows	3618	1717
HE (flow-based)	3440	1638
Rate (flow-based)	95.1%	95.4%
HE (packet-based)	3517	1676
Rate (packet-based)	97.2%	97.6%

3) Test 3: Non-encrypted Files: In this section we test our algorithms with non-encrypted (but not necessarily LE) files.

The goal is to ensure we don't see any anomalies. We carry out the experiment as follows: we use eight different types of files (based on file extension), ranging from MS Office files to ascii and popular media. We use a total of 150 files per type. We also ran tests with files compressed with five common compression tools, namely bzip, gzip, compress, Rar and Winzip. We used 30 each of .pdf, .doc, .xls, windows .TXT and Linux .txt files, for a total of 150 compressed files per tool. We include compressed files because we expect many of these to be HE. We again use N = 2 as *Sequential High Entropy Packets* and 90% as the *Entropy Packet Percentage Threshold*.

Table VII shows the results. We see that mostly, only a few of the media files are classified as HE. Happily, none of the ascii files is classified as HE by either algorithm. For the compressed files, as expected, many are classified as HE by both algorithms, with the interesting exception of .Z files.

Our conclusion is that the packet-based algorithm seems more eager to classify data as HE than the flow-based algorithm. This is not surprising, since the flow-based algorithm looks at the entropy of the entire data in a flow or file. There are a couple of surprises, however: (a) none of the .Z files are detected as HE, and (b) only the packet-based algorithm classifies at least some of the image and audio files as HE. It would be very interesting to dig deeper into these results, but this falls outside the scope of the current work.

TABLE VII
OFFLINE NON-ENCRYPTED FILES

File Type	HE (flow-based) out of 150	HE (packet-based) out of 150
DOC	2	11
EXCEL	0	6
Windows TXT	0	0
Linux text	0	0
PDF	4	15
JPG	0	33
MP3	0	42
Executable	13	15

TABLE VIII
OFFLINE NON-ENCRYPTED COMPRESSED FILES

File Type	HE (flow-based) out of 150	HE (packet-based) out of 150
Z	0	0
bz	11	61
gz	30	103
Rar	36	115
zip	29	107

VI. ENHANCING BOTHUNTER WITH A HIGH ENTROPY DETECTOR

Armed with two HE classification algorithms, we move toward enhancing BotHunter (BH) with an additional event that enables the detection the high-entropy flows. The hope is that the presence of HE flows in conjunction with existing bot events that BH already detects, will flag bots using encrypted traffic. In this section we first outline our modifications to BH, and then use our bot traces after we encrypt them to determine true positives and our lab traces to determine false positives.

We enhance BH as follows. First, we use our algorithms to implement a HE flow detector separate from BH (recall that we don't have access to BH's source code to integrate our code). Our detector adds an entry to the snort log when it detects at least one HE flow between two hosts. We then resort to a hack, where our detector triggers an existing event in BH by adding the appropriate entry in the snort log, namely the egg download event. Recall from our previous analysis that the egg download event has a weight of 0.5, which we deem as the upper bound for our HE detector. The reason is that the egg download event is triggered using DPI, which carries a much higher confidence than our detector. In our experiments we try weight values from 0.1 to 0.4.

For the HE flow detector, we chose the packet-based algorithm. While this algorithm tends to flag more flows as HE than its flow-based counterpart, we felt that BH will suppress many of the false positives. Moreover, we plan to use this detector on a 10G link, so speed is very important. While not done yet, we plan to repeat our experiments with the flow-based detector too.

We used all our available traces for this evaluation. The bot traces *140Samples* were used to detect true positives. We altered the traces by XOR'ing the payload of each packet during the infection phase, which renders DPI techniques in BH ineffective. Recall that the reason for using XOR is that it is a low barrier for an attacker seeking to evade BH. Note, however, that XOR does not necessarily change the flow entropy, so when our entropy detector flags flows as HE they may have already been HE before using XOR. Our conjecture is that if an attacker uses other encryption techniques, this can only increase the likelihood that our HE detectors will flag flows as HE. In some sense, using XOR represents not only the lowest barrier of entry for evasion but also the hardest case for HE detection.

The network traces *Lab1*, *Lab2* and *Lab3* are used to evaluate false positives. These traces were captured from our lab network and are not likely to contain any bots. We did run the unmodified BH on these traces to verify this assumption and found five suspicious machines. Four of them were included in our research projects. Two were running rDNSD, which performed DNS querying, one was a GENI rack machine and another is part of a project that periodically pings the entire Internet. After excluding these, the remaining host could be a bot, but it comes from our DHCP pool and we don't have ground truth for it.

We carry out two experiments to select the proper weight for the entropy detector. In the first experiment we measure true positives and in the second false positives. The results are shown in Figure 4 and Figure 3. The x-axis in both figures is the weight assigned to the HE detector and the y-axes are a count of true and false positives respectively.

In Figure 3, when the score of the HE detector is 0.4 or less we notice one potential false positive in *Lab1*. The reason is that events *C&C communication (nbr)* and *Outbound Scan*

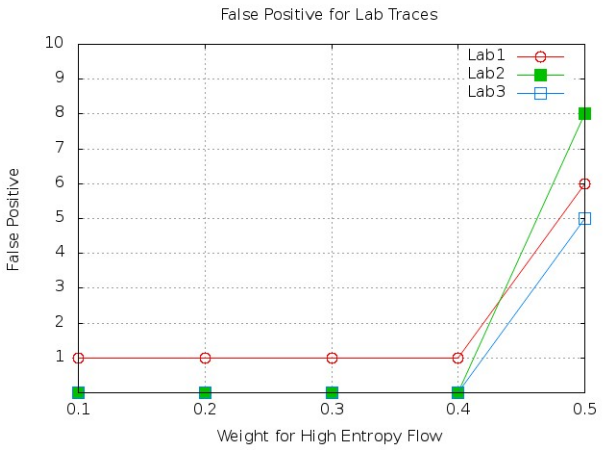

Fig. 3. False Positives for Lab Traces

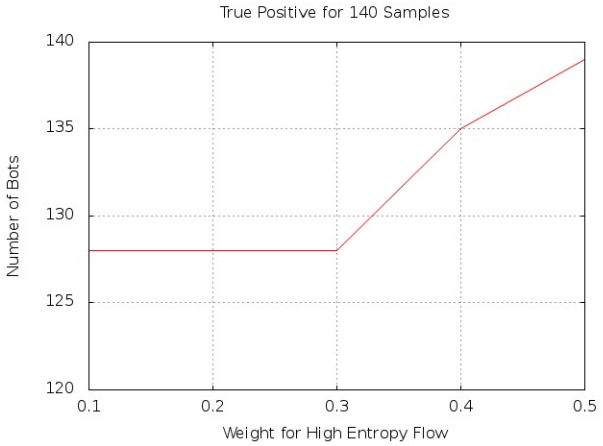

Fig. 4. True Positives for bot traces

based on Behavior are triggered, whose total score is 0.7. At the same time, this host also has high entropy connections, so the *HE detector* is triggered and the final score is greater than threshold. As a result, this host is flagged as a bot. This is the DHCP host we don't have ground truth for. Finally, Figure 4 shows that the enhanced BotHunter is able to detect 135 out of the 140 known bots when the weight of the detector is 0.4. We thus conclude that a weight value of 0.4 works best.

VII. Related Work

The first challenge in detecting encrypted botnets is to detect encrypted traffic and it can be done in two ways, host based [2] [16] and network based [5]. [16] investigates botnet detection based on automatic protocol reverse engineering; it can also deal with encrypted traffic by monitoring the memory buffer where the data is encrypted and decrypted. Compared to [16], [5] aims at detecting encrypted traffic by using real network traces and it can distinguish encrypted files from other high entropy files by using an index of coincidence; However, it requires large size data to make a decision because it may not be compressed as efficiently as small data. [8] utilizes entropy to detect encrypted and packed malware, but it only works for offline executable files. The work in [17] is related to ours, but the goal is to filter out opaque (compressed or encrypted) traffic in order to improve IDS performance.

VIII. Conclusions

In this paper we make an initial attempt to investigate detection of bots that use encrypted communication. We first show that encryption in botnet communication foils bot detection methods based on DPI. BotHunter relies on DPI and thus operates with reduced efficiency in the presence of encryption. However, encryption increases entropy. We build high-entropy classifiers and add them to BotHunter. Enhanced with our high-entropy detectors, we showed that BotHunter is now able to detect even encrypted bots.

References

[1] Bothunter dialog event. http://www.bothunter.net/about.html.

[2] J. Caballero, P. Poosankam, D. Song, and C. Kreibich. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *In CCS09: of the 16th ACM conference on Computer and communications security*, pages 621–634. ACM, 2009.

[3] K. Chiang and L. Lloyd. A case study of the rustock rootkit and spam bot. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, pages 10–10, Berkeley, CA, USA, 2007. USENIX Association.

[4] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware analysis via hardware virtualization extensions. In *In Proceedings of the 15th ACM Conference on Computer and Communications Security*, pages 51–62, 2008.

[5] P. Dorfinger, G. Panholzer, and W. John. Entropy estimation for real-time encrypted traffic identification. In *Proceedings of the Third international conference on Traffic monitoring and analysis*, TMA'11, pages 164–171, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[7] 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 *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 9:1–9:9, Berkeley, CA, USA, 2008. USENIX Association.

[8] R. Lyda and J. Hamrock. Using entropy analysis to find encrypted and packed malware. *IEEE Security and Privacy*, 5(2):40–45, Mar. 2007.

[9] J. Olivain and J. Goubault-Larrecq. Detecting subverted cryptographic protocols by entropy checking. Technical report, Laboratoire Spcification et Vrification, June 2006.

[10] P. Porras, H. Sadi, V. Yegneswaran, P. Porras, H. Sadi, and V. Yegneswaran. A multi-perspective analysis of the storm (peacomm) worm. available at: http://www.cyber-ta.org/pubs/stormworm/report, 2007.

[11] P. Royal. Analysis of the kraken botnet, 2008.

[12] J. H. S. Stover, D. Dittrich and S. Deitrich. Analysis of the storm and nugache trojans - p2p is here. *Login, USENIX*, 32(6), December, 2007.

[13] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27:379–423, July 1948.

[14] E. Stinson and J. C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *Proceedings of the 2nd conference on USENIX Workshop on offensive technologies*, pages 5:1–5:9, Berkeley, CA, USA, 2008. USENIX Association.

[15] P. Wang, S. Sparks, and C. C. Zou. An advanced hybrid peer-to-peer botnet. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association.

[16] Z. Wang, X. Jiang, W. Cui, X. Wang, and M. Grace. Reformat: Automatic reverse engineering of encrypted messages. In M. Backes and P. Ning, editors, *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 200–215. Springer, 2009.

[17] A. M. White, S. Krishnan, P. Porras, M. Bailey, and F. Monrose. Clear and present data: Opaque traffic and its security implications for the future. In *NDSS*, 2013.