Detection and Classification of Different Botnet C&C Channels

Gregory Fedynyshyn[1], Mooi Choo Chuah[2], and Gang Tan[2] *

[1] Lehigh University. Bethlehem, PA 18015, USA,
gef209@lehigh.edu
[2] {chuah, gtan}@cse.lehigh.edu

Abstract. Unlike other types of malware, botnets are characterized by their command and control (C&C) channels, through which a central authority, the *botmaster*, may use the infected computer to carry out malicious activities. Given the damage botnets are capable of causing, detection and mitigation of botnet threats are imperative. In this paper, we present a host-based method for detecting and differentiating different types of botnet infections based on their C&C styles, e.g., IRC-based, HTTP-based, or peer-to-peer (P2P) based. Our ability to detect and classify botnet C&C channels shows that there is an inherent similarity in C&C structures for different types of bots and that the network characteristics of botnet C&C traffic is inherently different from legitimate network traffic. The best performance of our detection system has an overall accuracy of 0.929 and a false positive rate of 0.078.

Keywords: botnet detection, network security, host-based intrusion detection system

1 Introduction

Botnets are organized networks of infected (zombie) machines running bot code, categorized by their use of a command and control (C&C) channel. Through the C&C channel, a central authority (i.e., the *botmaster*) may issue commands to his army of zombie machines and essentially take full control over the infected machines. These networked armies of zombie machines are typically used to carry out an array of malicious activities, including, but not limited to, engaging in spam campaigns, stealing personal or financial information, participating in click-fraud campaigns, initiating distributed denial of service (DDoS) attacks, and propagating bot code to other vulnerable machines. Due to the vast amounts of damage botnets can cause, detecting and mitigating infections are imperative.

* The authors would like to thank Alex Lanstein of FireEye for all around, general assistance, and Dezhao Song and Xu Li of Lehigh University for help in collecting some botnet data.

1.1 Background: C&C Channel Types and Fast Flux

Current botnet C&C channels follow a general model: first, a botmaster must issue a command to the botnet; second, the botnet performs activities in response to the command; and third, the botnet sends the results of performing its activities back to the botmaster. There are 3 types of C&C channels, namely (a) Internet Relay Chat(IRC)-Based C&C channels which use a *push-based* model, where the botmaster *pushes* new commands to the botnet, which then responds directly to the commands, (b) HTTP-based C&C channels which use a *pull-based* model where bots periodically poll the C&C server to request new commands, (c) Peer-to-peer (P2P) based C&C channels where peer-to-peer communication is used to proxy commands or to locate a C&C server. P2P-based C&C has the advantage of not having a single point of failure which is inherent to IRC-based and HTTP-based bots.

Many botnets use a DNS technique called *fast flux* to hide their central C&C server. The idea behind *fast flux* is to use an ever-changing array of compromised hosts to proxy messages between the central C&C server and the botnet. The botmaster continually changes the proxy to which a domain name points and bots find the C&C proxy by looking up the domain name instead of using hard-coded, static IP addresses. In the event a proxy is taken down, the central C&C server will remain intact and continue to issue commands through different proxies, adding a layer of resiliency and stealth to the botnet.

1.2 Botnet Detection Approaches

Most botnet intrusion detection systems (IDS) fall into three categories: host-based, network-based, and a hybrid of the two. Host-based systems, such as [1–3] focus on detecting bot infections on an individual host and typically use signature- or behavior-based methods to correlate network traffic or system events with known bot signatures or behavioral information. While host-based IDS's are able to detect single bot infections, some knowledge of the bot's behavior must be known in advance. Host-based approaches also benefit from being easy to deploy and from empowering the end-user directly.

Network-based methods, such as [4–7], attempt to detect bot infections by correlating similar behaviors among several different hosts on the monitored network. Network-based methods do not need prior knowledge of bot signatures or behavioral information as they rely on the intuition that hosts infected by the same bot will behave very similarly to one another whereas uninfected hosts will exhibit different network characteristics from one another. While network-based intrusion detection systems (IDS) may not require prior knowledge of a bot's behavioral patterns, they do require that multiple hosts in the same network become infected for the intrusion to be detectable. In addition, network-based approaches may require additional cooperation of the network administrator and care must be taken to protect the privacy of the network users.

1.3 Contributions

We adopt a host-based IDS in this paper. Our contributions to the field are:

1. Our system detects not only bot infections independent of packet payload content, but also types of C&C channels, as knowing the type helps with deploying appropriate defenses. To identify botnets and their types, our system identifies *persistent* connections (similar to [3]), however, unlike [3] our system can distinguish the type of C&C channel.
2. A domain-based approach is used to undermine the effectiveness of fast flux obfuscation techniques. The domain-based approach groups conversations using full domain names rather than IP addresses, allowing us to deobfuscate fast flux botnets.
3. A binary classifier to identify IRC-based traffic instead of examining packet payload content (which consumes much processing power and is not viable for encrypted IRC traffic) or using popular IRC port numbers (which may miss IRC traffic on non-popular IRC ports).

Our preliminary evaluation shows that there is a similarity inherent in the traffic produced in botnet C&C communications that is different from legitimate network traffic. Additionally, we show that each C&C style shares similarity across multiple botnet families. This inherent similarity implies that that, unlike many host-based IDS's, our model has the potential to discover infections of previously unknown bots.

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Section 3, we discuss hypotheses, objectives, and the architecture of our IDS framework. In Section 4, we present the evaluation of our approach with the results presented in section 5. In Section 6, we provide some discussions of why our approach works and in Section 7, we conclude by discussing work that we intend to carry out in the near future.

2 Related Work

Much work has been done on botnet intrusion detection. Many IDS's make use of the observation that the characteristics of botnet network traffic differ greatly from those of normal network traffic. Many botnet detection systems rely on *anomaly detection* to discover bot infections. Anomaly detection simply aims to detect significant deviations from normal behavior and is typically applied to network behavior, such as the characteristics of network connections, or system-level behavior, such as CPU usage or modifications to the file system. The main advantage of anomaly detection is that it is good at discovering new infections, as infections are likely to cause changes in the monitored activity no matter what shape they take.

There are different types of botnet detection methods, e.g. host-based or network-based systems. The host-based IDS presented in [1] uses anomaly detection on aggregate network features to identify a deviation from normal activity.

Once identified, a snapshot of the network traffic surrounding the anomaly is taken. Using the intuition that snapshots containing similar anomalies are likely multiple instances of a bot responding to the same botmaster command, the packet payloads leading up to the anomaly are searched for common content to find the command. Once a suitable representation of the command is found, the IDS can build a profile which can then be used to detect future occurrences of the command/response pair. One drawback to this IDS is its need to examine packet payloads. Many current-day botnets use encryption to obfuscate their C&C messages. If the botnet uses a complex encryption algorithm for its C&C channel, this approach will not find any commonality in packet payloads containing botmaster commands.

Another host-based IDS presented by Giroire et al. [3] is based on the intuition that bots must contact their C&C server regularly to receive commands from the botmaster. Thus, unlike transient connections, the connections to C&C channels will appear to be *persistent*. This IDS first builds a whitelist of legitimate destinations that the monitored host contacts persistently. If any new connection is observed that exhibits high enough temporal persistence, an alarm is raised. If this connection is legitimate, a user can simply add it to the whitelist, otherwise, the connection is assumed to be malicious and is blocked. The success of such a system relies on the assumption that the whitelist is easy to maintain and that it does not need to be updated frequently. The system presented in this paper also uses the notion of observing persistent connections to detect botnet C&C channels.

The network-based IDS, BotSniffer [5] exploits the *spacial* and *temporal* similarity of botnet activity to differentiate between botnet network traffic and legitimate network traffic. The inspiration behind BotSniffer is that each bot in the network will receive a similar or identical command at similar times, and then perform similar activities in response to the command at similar times. Potential C&C messages are limited to incoming IRC and HTTP packets. Bot responses are categorized into two types: *message responses*, such as sending a message to an IRC chat room, and *activity responses*, such as scanning the network or sending spam emails. Botnet detection works as follows: if at any point in time a group of hosts is observed to be performing similar activities in response to similar messages from the same server, they likely belong to the same botnet.

Similar to BotSniffer, BotMiner is a network-based IDS [6]. BotMiner categorizes network activity into *communication activity* that corresponds to potential C&C communication and *malicious activity* that corresponds to scanning, spamming, or binary download events. BotMiner clusters hosts according to similar communication activities and according to similar malicious activities, then performs cross-cluster analysis to identify hosts that share both similar communication and malicious activities. While the BotMiner IDS had impressive results, it still falls prey to the same problem most network-based detection schemes do: that multiple hosts in the monitored network must be infected to be detected by BotMiner.

3 Methodology

The goal of our host-based IDS is to be able to detect the presence of botnet C&C traffic on the monitored host, as well as classify the style of C&C communication the bot is using, be it IRC, HTTP, or P2P. Furthermore, our detection system is completely independent of the content of the C&C messages, i.e., we do not examine packet payloads. The ability to locate and classify botnet C&C connections depends on a few hypotheses:

1. Botnet C&C communication can be differentiated from botnet non-C&C communication.
2. Botnet C&C communication can be differentiated from legitimate communication.
3. The characteristics of different styles of C&C are similar across different botnet families.

We will use the term *conversation* to refer to all network packets transmitted between two unique hosts, based on full domain name when available, otherwise IP address. We use full domain name to undermine fast flux techniques. As a botnet C&C channel may use several different servers in a fast flux network to perform the same service of issuing commands to the bot, we wish to capture the full C&C conversation, which an IP addressed based approach would fail to do. Unless otherwise noted, we will refer to a *conversation* as being between the monitored host and some destination host. We will also present the results when IP addresses are used rather than domain names to define conversations at the end of Section 5 for comparison. We use the observation that while humans and bots alike may connect to a vast quantity of different destination addresses during the course of any given day, the number of connections that exhibit long-term, continual two-way traffic is relatively small. We also use the observation that a bot must be in continual contact with its botmaster to effectively be a member of a botnet. Thus, the C&C channel must appear as a conversation that exhibits continual two-way network traffic over the course of time.

Instead of examining features of aggregate network activity on the monitored host, we examine features on a per-conversation basis. We begin by dividing the network traffic for each conversation into time slots of length t. Within each time slot, we track the total number of bytes and packets sent, the total number of bytes and packets received, the protocol and the ports used in the conversation. We then generate *instances* with which to train the classifier by examining statistical features contained within an *observation window* that is n time slots in length. Thus, the network features as calculated across the time slots in each observation window generate a single *instance* of a conversation. Note that a conversation that lasts longer than n time slots will generate more than one instance. Let W be the observation window such that $W = \{s_i, s_{i+1}, ..., s_{i+n-1}\}$ where s_i is the network traffic contained in time slot i. Figure 1 shows an example of the relationship between time slots to the observation window where the observation window consists of twelve time slots.

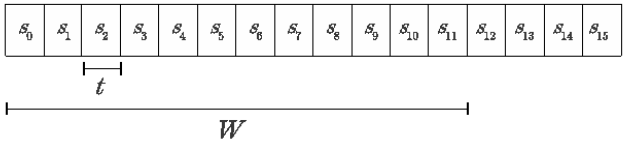

Fig. 1. Example showing time slots and observation window

3.1 Data Collection

In this section, we describe the data collection process. As we will eventually be training a classifier, we need a good set of network traces which are comprised of legitimate traffic as well as a set of network traces comprised of botnet traffic.

Normal Traces: To collect data with legitimate network traffic, we used a libpcap-based packet capture program to capture all network traffic on four hosts for a combined total of 29 days. We do not want to promiscuously capture network traffic for two reasons. First, we are implementing a host-based IDS, so we do not want to look at the network activity of other hosts on the network. Second, we know that the machines used to generate normal network activity are clean of malware but cannot be certain about other hosts on the network. An IDS is useless if it is not robust enough to work on any given host. To that end, we ensured our normal traces covered a wide range of legitimate activity, such as instant messaging, email communications, web browsing, video streaming, SSH, IRC, etc.

Botnet Traces: Collecting botnet traffic is a more challenging affair. Our requirements for running bot samples are: to make sure that the generated traffic is typical of the specific bot and to make sure the bot does not damage any other hosts on the network. To tackle the first requirement, we ran our bot samples on a virtual machine running Windows XP SP2 and recorded network traffic using a libpcap-based capture program. Virtual machines can be easily reverted back to their pre-infection state, which established a clean baseline from which to run the bot samples. As the host machine running the virtual Windows XP system is on the Lehigh University campus network, it is important to make sure it will not cause any damage to the network. For example, many botnets infect new users through worms and OS exploits, so outgoing traffic on known exploit ports was blocked. We believe that our measures were sufficient in avoiding any damage to the Lehigh University campus network. Table 1 shows a summary of the network data collected. In many cases, data for botnet families were generated using different variants of the bot. Botnet family identification was performed using the open source ClamAV antivirus tool [8].

Table 1. Summary of data collected

Trace Type	C&C Type	Number of Variants	Total Length
Normal	NA	NA	29 days
Ircbot	IRC	4	8 days
Agobot	IRC	3	18 days
Rustock	HTTP	2	5 days
Storm	HTTP	4	17 days
Bobax	HTTP	4	5 days
Waledac	P2P	2	6 days
UDP Storm	P2P	1	6 days

3.2 Data Processing

We define *persistence* in terms of the number of time slots in the observation window in which two-way communication occurs divided by the total number of time slots in the observation window. For example, if two-way communication is only observed in half of the time slots in the observation window, the *persistence* of the conversation in that particular observation window is 0.5.

Figure 2 provides a graphical representation of the steps taken to create instances which can then be sent to the Botnet Classifier for botnet detection:

1. Collect network traffic
2. Split network trace into conversations
3. Divide conversations into observation windows and extract feature values to create instances
4. Filter out impersistent instances
5. Pass final instance set to Botnet Classifier

Only instances that pass the persistence test are retained. For each retained instance, we compute the following features across the time slots contained within the instance:

- Standard deviation of bytes sent / byte received
- Standard deviation of packets sent / packets received
- Standard deviation of bytes sent / packets sent
- Standard deviation of bytes received / packets received
- Standard deviation of the number of source ports used
- Standard deviation of the number of destination ports used
- Average number of bytes sent
- Average number of bytes received
- Average number of packets sent
- Average number of packets received
- Average number of TCP packets transmitted
- Average number of UDP packets transmitted

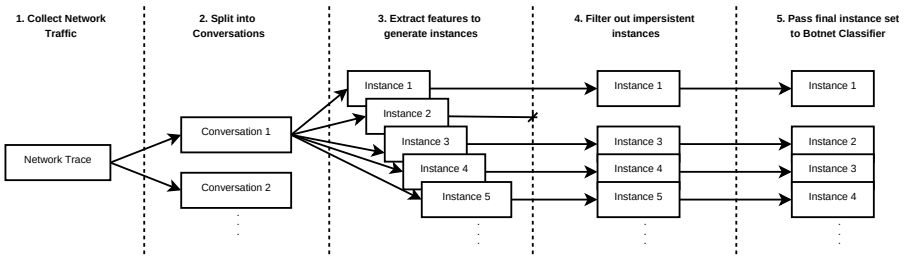

Fig. 2. Instance generation steps

– Average number of IRC packets transmitted
– Persistence value

Furthermore, let p be the *persistence threshold*. If the persistence value calculated for an instance of a conversation is less than the *persistence threshold*, the instance is considered to be transient, and therefore not a likely candidate for a potential C&C channel. A conversation that persists over time may produce some instances that do not exceed the persistence threshold. Such a situation may arise from a user shutting down her machine or losing her Internet connection. Thus, we decide to filter out these impersistent instances as they do not accurately reflect the true behavior of the conversation that our botnet classifier is concerned with.

Through experimentation, we found that $t = 600$ seconds (10 minutes), $n = 24$ time slots (4 hours), and $p = 0.6$ produces the best results. As our data clearly spans more than 4 hours, we can make use of a sliding observation window to generate multiple instances for a single conversation. Defining instances based on observation windows has some benefits. Conversations can theoretically persist indefinitely. Instead of having to store statistics regarding entire conversations in computer memory, we only need to keep track of a handful of observation windows. In addition, were there a situation where a legitimate site becomes hijacked and used for malicious purposes, the observation window would soon slide past the time slots encompassing the legitimate traffic and be able to detect the new, malicious behavior.

There are two approaches we can take when determining how far to slide the observation window, W, to generate the next instance: slide W a single time slot or slide W n time slots, such that no observation windows overlap. For an online IDS, it would make the most sense to slide W by one time slot at a time as overall detection time would decrease. On the other hand, sliding W by n time slots at a time ensures that there is no overlap from one instance to the next. Overlapping instances may skew the accuracy of the classifier favorably, so we present results based on a non-overlapping sliding observation window. Using a non-overlapping observation window with $t = 600$ seconds and $n = 24$ time slots, a 24-hour long conversation can generate at most 6 instances. Using an overlapping observation window with the same values for t and n, a 24-hour

long conversation can generate at most 20 instances.

Table 2 shows the number of instances generated from the persistent conversations using the domain-based approach. Note the relatively low number of persistent conversations found in each trace compared to the overall number of conversations. As Table 2 shows, the number of persistent conversations in both normal and botnet traces is immensely smaller than the total number of conversations. By filtering out all transient conversations, we are left only with likely C&C conversations. In addition, by disregarding impersistent connections, we drastically cut down the amount of network traffic we need to analyze, leading to a more computationally efficient model.

During our evaluation, we split our 89 total days network traffic such that half (44 days) is used to build the training set and the other half (45 days) is used to build the testing set, where there is no overlap between the training and testing data. The number of instances generated for both training and testing are also shown in Table 2.

Table 2. Summary of instance generation

Trace Type	Total Convs	Persistent Convs	Training Instances	Testing Instances
Normal	8,662	98	95	128
Ircbot	377	31	57	89
Agobot	32	4	17	10
Rustock	181	7	15	21
Storm	433	18	12	155
Bobax	42,307	24	40	51
Waledac	705	16	60	59
UDP Storm	1341	126	116	194

3.3 IRC Binary Classifier

Calculating the value of the average number IRC packets requires that we know which conversations are IRC sessions and which are not. While typical IRC servers are run on ports in the 6667-6669 range, many botnets use other ports for IRC servers as a way to obfuscate their presence. We did indeed notice this behavior with several of the bot samples we ran, with one contacting an IRC server on port 65520. One can examine packet payload content for strings common to IRC sessions, however any approach that examines packet payload content would fail if the IRC session is encrypted. Thus, we need a scheme for detecting IRC sessions that satisfies the following criteria:

1. Must be port independent
2. Must be packet payload-content independent

3. Must be accurate

Our solution was to build a binary classifier to determine whether a conversation is an IRC session or not. The IRC binary classifier should not be confused with the botnet classifier, as its aim is to distinguish IRC sessions from non-IRC session regardless of whether they belong to a bot or a legitimate user. To build the IRC classifier:

1. Make a copy of our training and testing sets
2. Manually examined our network traces to locate IRC conversations
3. Manually remove encrypted conversations (i.e., unsure if IRC)
4. Set the class label of the instances corresponding to IRC conversations to be "IRC"
5. Set the class label of the instances not corresponding to IRC conversations to be "Non-IRC"

While manually examining network traces to locate IRC conversations is an expensive process, it only needs to be done once to build an appropriate training set for building the classifier. We found a J48 classifier to perform best at detecting IRC sessions. We performed sensitivity analysis on our full feature set to find a reduced feature set for the IRC classifier that gave the best performance:

- Standard deviation of bytes received / packets received
- Standard deviation of the number of source ports used
- Average number of bytes received
- Average number of packets sent
- Average number of packets received
- Average number of UDP packets transmitted

A single conversation can generate several instances according to our scheme, meaning that it is possible that our IRC binary classifier could classify some instances of a conversation as being IRC traffic and some instances as not being IRC traffic. Such a result is not reflected in reality as IRC sessions will continue to be IRC sessions. Rather than labeling single instances, we decided to label entire conversations as being IRC or not. Our approach begins by classifying the instances in each conversation as being instances of IRC traffic or non-IRC traffic. If the majority of instances in a conversation are labeled as being IRC traffic, we consider the entire conversation to be an IRC session and update all of its instances to be considered IRC instances. Similarly, if the majority of instances in a conversation are labeled as non-IRC instances, the entire conversation is considered not to be an IRC session and all of its instances are updated to reflect this. Figure 3 illustrates our IRC binary classification process. The results of our IRC binary classifier are presented in Table 3. For the IRC binary classifier, accuracy is defined as the number of correctly classified conversations divided by the total number of conversations. Using the IRC binary classifier, we are able to differentiate IRC sessions from non-IRC sessions with an accuracy of 0.977. Now that we can be fairly confident about which conversations are IRC sessions, we are able to determine the value of the "average number of IRC packets" feature.

Table 3. Results of IRC Classifier on conversations

Classified as →	NON-IRC	IRC
Non-IRC	151	2
IRC	2	17

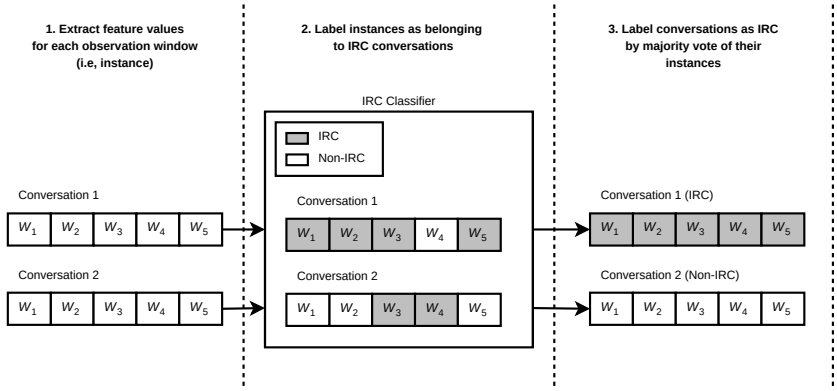

Fig. 3. Classification of IRC conversations: domain-based

4 Evaluation

We used the Weka Machine Learning Java library [9] to build our classifiers. During the training phase, all generated instances (see Table 2) are labeled with one of four class values indicating the type of C&C channel used: *NORMAL*, *IRC*, *HTTP*, *P2P*. Labeling is based on prior knowledge of the bots used to generate network traces. We examined the effectiveness of the two best performing classifiers, a J48 classifier and a Random Forest classifier [9]. As there was no overlap in destination hosts for persistent conversations between normal and botnet traces, testing the classifier on the generated test instances is equivalent to overlaying botnet traces on top of normal traces because the data processing stage would separate the individual conversations from the merged traces.

In addition, we performed sensitivity analysis on our feature set to select reduced subsets of features that produced the best overall accuracy. Trying all possible combinations of features from our total fourteen-feature set, the best set of features for the J48 classifier was found to be:

- Standard deviation of packets sent / packets received
- Average number of bytes sent
- Average number of bytes received
- Average number of TCP packets transmitted
- Average number of UDP packets transmitted
- Average number of IRC packets transmitted

– Persistence value

Similarly, the best feature set for the Random Forest classifier was found to be:

– Standard deviation of bytes sent / bytes received
– Standard deviation of the number of source ports used
– Average number of bytes sent
– Average number of bytes received
– Average number of packets sent
– Average number of UDP packets transmitted
– Average number of IRC packets transmitted
– Persistence value

5 Results

Accuracy is defined as the number of instances correctly classified divided by the total number of instances. False positive rate is defined as the number of instances of legitimate traffic classified as botnet traffic divided by the total number of instances of legitimate traffic. The results of running the classifiers on the reduced feature sets are shown for both the J48 classifier and the Random Forest classifier in Table 4. The J48 classifier had an overall accuracy of 0.926 with a false positive rate of 0.188. The Random Forest classifier had an overall accuracy of 0.929 with a false positive rate of 0.078.

Table 4. Results of Botnet Classifiers: domain-based

	J48				Random Forest			
Classified as →	NORMAL	IRC	HTTP	P2P	NORMAL	IRC	HTTP	P2P
Normal	104	0	21	3	118	0	5	5
IRC	0	99	0	0	0	99	0	0
HTTP	21	1	201	4	32	1	189	5
P2P	2	0	0	251	2	0	0	251

In the results we presented in Table 4, we chose to classify individual instances of conversations rather than entire conversations since we are hoping to produce an online detection system eventually. With an observation window of 4 hours long, our online IDS would potentially be able to detect the presence of a bot within 4 hours of the initial infection. However, it is more intuitive to think of botnet C&C detection in terms of detecting entire C&C conversations. Thus, we also report the accuracy rate of our detection method for classifying entire conversations using a majority vote of the classification results for the instances generated from a conversation. Our results for both the domain-based approach

as well as the IP-based approach are shown in Table 5. On a per-conversation basis, accuracy is defined as the number of correctly classified conversations divided by the total number of conversations. False positive rate is defined as the total number of normal conversations classified as botnet conversations divided by the total number of normal conversations. For the domain-based approach, the J48 classifier correctly classified 169 out of a total of 181 conversations and the Random Forest classifier correctly classified 171 of 181 conversations. On a per-conversation level, the J48 classifier had an accuracy of 0.934 and a false positive rate of 0.173 and the Random Forest classifier had an accuracy of 0.945 and a false positive rate of 0.038.

Whereas the domain-based approach found a total of 19 IRC botnet conversations, the IP-based approach found 30. However, in many cases, multiple IP addresses of IRC servers corresponded to a single domain name. Though only 19 persistent connections to domain names were found in the domain-based approach for IRC-based botnets, they accounted for a total of 48 IP addresses, meaning that the IP-based approach missed 18 of the IP addresses associated with the IRC C&C channels. Thus, the results presented in Table 5 are adjusted to include the missed IP addresses. The IP-based approach only found 30 of 38 IP addresses associated with HTTP C&C channels. Both the P2P-based bots did not demonstrate multiple IP addresses associated with a single domain name. While the IP-based approach missed 8 IP addresses associated with persistent conversations to domain names for the instances generated from legitimate traffic, it does not make sense to count those additional IP addresses as misclassified when calculating false positives, as a missed conversation can not raise an alarm. Therefore, the number of normal conversations found by the IP-based approach is left at 48 in Table 5. The per-conversation accuracy of the J48 classifier using the IP-based approach is 0.752 with a false positive rate of 0.200 and the accuracy of the Random Forest classifier using the IP-based approach is 0.771 with a false positive rate of 0.042. Compared to the per-conversational accuracy and false positive rate achieved using the domain-based approach, it is clear that the IP-based approach has poorer performance.

Table 5. Results of Botnet Classifiers (entire conversations)

	Domain-based				IP-based			
	J48		Random Forest		J48		Random Forest	
Conversations	Total	Correct	Total	Correct	Total	Correct	Total	Correct
Normal	52	43	52	50	48	40	48	46
IRC	19	19	19	19	48	24	48	23
HTTP	34	31	34	26	38	24	38	23
P2P	76	76	76	76	76	76	76	76
Total	181	169	181	171	218	164	218	168

6 Discussion

We describe three hypotheses at the beginning of Section III that have to be true in order for our approach to work. The results we have presented in the previous section have shown that botnet C&C communication can be differentiated from botnet non-C&C traffic through the use of a persistence metric. The intuition that botnet traffic follows an inherent command-response pattern such that it can be differentiated from normal, legitimate traffic was shown to be true, as our classifiers were able to successfully distinguish between normal and botnet traffic. Furthermore, the thought that the characteristics of C&C styles across different botnet families would still contain inherent similarities was also shown to be true, as both of our classifiers were able to successfully differentiate the different C&C styles across multiple variants of bots in the same botnet family, and across bots from different bot families. We have also shown that, while IP-based approaches to botnet detection may appear to produce decent results, they run both the risk of missing connections to malicious IP addresses as well as the risk of not capturing the entire behavior of a C&C conversation in the presence of fast flux DNS techniques.

Ultimately, the Random Forest classifier produced the best results in terms of accuracy and false positive rate. For an online IDS, one could either prompt the user when a suspicious conversation is detected, asking her whether or not she wants to block the connection or could automatically block the connection. A high false positive rate would either annoy the user by raising prompts too frequently or would block legitimate connections, leading to further user annoyance. Thus, minimizing the false positive rate is imperative. Sometimes, a botnet conversation may not be detected when it first appears in an observation window but it is very likely that it will be detected in subsequent windows. We hope to quantify the detection time in the near future. As a single, persistent conversation will generate multiple classifiable instances as time progresses, even if the bot infection is not detected in the first observation window, it could certainly be detected in a future observation window. Furthermore, many bot samples initiated several persistent conversations. To successfully detect a bot infection, we only really need to discover one of the persistent conversations. To this end, our IDS was able to detect every bot infection even if it misclassified some of the instances generated from the multitude of bot conversations.

7 Concluding Remarks

Botnets are serious threats to computer networks. Malicious activities such as sending spam, stealing personal or financial information etc can be launched by botnets. In this paper, we present a host-based method for detecting and differentiating different types of botnet infections e.g. IRC-based, HTTP-based or P2P based bots. Our method includes a few unique features, namely (a) the ability to correctly identify the C&C style, (b) a binary IRC classifier that allows identification of IRC traffic without payload inspection, and (c) a domain-based approach that helps to deal with fast flux obfuscation techniques which

are becoming more popular in botnet traffic that has been identified in recent months. Our detection scheme can achieve an accuracy of 0.929 with a false positive rate of 0.078 and a false negative rate of 0.033 using the Random Forest classifier. In the near future, we would like to extend our work to detect botnet infections on mobile devices as the growing popularity of smartphones is making them a growing target for hackers to exploit.

References

1. Peter Wurzinger and Leyla Bilge. *Automatically Generating Models for Botnet Detection*, European Symposium on Research in Computer Security, 2009.
2. Liberios Vokorokos, Anton Balaz, Martin Chovanec. *Intrusion Detection System Using Self Organizing Map*, Acta Electrotechnica et Informatica, 2006.
3. Frederic Giroire, Jaideep Chandrashekar, Nina Taft, Eve Schooler, Dina Papagiannaki. *Exploiting Temporal Persistence to Detect Covert Botnet Channels*, Recent Advances in Intrusion Detection, 2009.
4. Anirudh Ramachandran, Yogesh Mundada, Mukarram Bin Tariq, Nick Feamster. *Securing Enterprise Networks Using Traffic Tainting*, Special Interest Group on Data Communication, 2008.
5. Guofei Gu, Junjie Zhang, Wenke Lee. *BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic*, Network and Distributed System Security, 2007.
6. Guofei Gu, Roberto Perdisci, Junjie Zhang, Wenke Lee. *BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection*, Proceedings of the 17th conference on Security symposium, 2008.
7. Su Chang and Thomas Daniels. *P2P Botnet Detection using Behavior Clustering & Statistical Tests*
8. Clam AntiVirus. http://www.clamav.net.
9. Weka 3 Data Mining and Machine Learning Software. http://www.cs.waikato.ac.nz/ml/weka/.
10. John John, Alexander Moshchuk, Steven Gribble, Arvind Krishnamurthy. *Studying Spamming Botnets Using Botlab*, Network Systems Design and Implementation, 2009.
11. Yuanyuan Zeng, Xin Hu, Kang Shin. *Detection of Botnets Using Combined Host- and Network-Level Information*, International Conference on Dependable Systems & Networks, 2008.
12. Joe Stewart. *Inside the Storm: Protocols and Encryption of the Storm Botnet*, http://www.blackhat.com/presentations/bh-usa-08/Stewart/BH_US_08_Stewart_Protocols_of_the_Storm.pdf, 2008.
13. Andreas Pitsillidis, Kirill Levchenko, Chritian Kreibich, Chris Kanich, Goeffrey Voelker, Vern Paxson, Nicholas Weaver, Stefan Savage. *Botnet Judo: Fighting Spam with Itself*, Network and Distributed System Security, 2009.
14. Phillip Porras, Hassen Saidi, Vinod Yegneswaran. *A Multi-perspective Analysis of the Storm (Peacomm) Worm* http://www.cyber-ta.org/pubs/StormWorm/report, 2007.
15. Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, Felix Freiling. *Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm*, USENIX Workshop on Large-Scale Exploits and Emergent Threats, 2008.