Peer to Peer Botnet Detection Based on Flow Intervals

David Zhao[1], Issa Traore[1], Ali Ghorbani[2], Bassam Sayed[1], Sherif Saad[1], and Wei Lu[3]

[1] Department of Electrical and Computer Engineering
University of Victoria
davidzhao@ieee.org, {itraore,bassam,shsaad}@ece.uvic.ca
[2] University of New Brunswick
ghorbani@unb.ca
[3] Keene State College
wlu@keene.edu

Abstract. Botnets are becoming the predominant threat on the Internet today and is the primary vector for carrying out attacks against organizations and individuals. Botnets have been used in a variety of cybercrime, from click-fraud to DDOS attacks to the generation of spam. In this paper we propose an approach to detect botnet activity by classifying network traffic behavior using machine learning classification techniques. We study the feasibility of detecting botnet activity without having seen a complete network flow by classifying behavior based on time intervals and we examine the performance of two popular classification techniques with respect to this data. Using existing datasets, we show experimentally that it is possible to identify the presence of botnet activity with high accuracy even with very small time windows, though there are some limitations to the approach based on the selection of attributes.

Keywords: Botnet, Network Intrusion Detection, Traffic Behavior Analysis, Network Flows.

1 Introduction

A bot is an autonomously operating software agent which may be controlled by a remote operator (the botmaster) to perform malicious tasks typically installed onto a victim's computer without the owner's consent or knowledge. Bots allow a remote operator to control the system and command it to perform specific, typically malicious tasks. Some of the tasks performed by a botnet include distributed denial of service (DDOS), mass spam, click fraud, as well as password cracking via distributed computing and other forms of cybercrime.

Command and control (C&C) is the key identifying characteristic of a botnet, and as such there is a variety of methods used by bots to form this network structure. Command and control channels must allow a botmaster to issue orders to individual bots in an efficient manner while at the same time avoiding being detected by computer security measures. Additionally, command and control channels would ideally want to be decentralized so that individual members are difficult to detect even if the C&C channel is compromised, allowing for resiliency in the network. As with

D. Gritzalis, S. Furnell, and M. Theoharidou (Eds.): SEC 2012, IFIP AICT 376, pp. 87–102, 2012.
© IFIP International Federation for Information Processing 2012

any problem, these three traits are frequently at odds with each other and botmasters and bots designers must make a tradeoff between stealthiness, decentralization and efficiency.

One of the most popular methods for implementing botnet command and control is by using the Internet Relay Chat (IRC) protocol [1]. IRC based C&C channels are highly efficient due to the ease of implementation as well as the capability of forming large networks very quickly due to the simplicity of the network architecture. Their weakness lies in their highly centralized nature: a compromise of a botnet C&C server may compromise the location of all bots connected to that server. Additionally, monitoring of network traffic may easily reveal the messages being passed from the server to individual clients, and much research has been done on botnet detection based on the analysis of these message contents.

C&C schemes based on HTTP traffic is another popular method for botnets. As a well-known protocol, HTTP based botnet C&C attempts to be stealthy by hijacking a legitimate communications channel in order to bypass traditional firewall based security and packets are often encrypted to avoid detection based on deep packet analysis. However, HTTP based C&C schemes still suffer from the issue of centralization, and it is possible to exploit such centralized behavior in their detection.

A more recent development in botnet C&C technology utilizes peer to peer (p2p) networks and protocols to form the communications network for bots. In p2p schemes, individual bots act as both client and server, producing a network architecture without a centralized point which may be incapacitated. The network is resilient in that when nodes are taken offline, these gaps may be closed automatically, allowing for the network to continue to operate under the attacker's control. [2] [3].

Feily et al. [4] divides the life-cycle of a botnet into five distinct phases: initial infection, secondary injection, connection, malicious command and control, and update and maintenance. In the initial infection phase, an attacker exploits a known vulnerability for a target system and infects the victim machine, granting additional capabilities to the attacker on the target system. In the secondary injection phase, the attacker uses his newly acquired access to execute additional scripts or programs which then fetch a malicious binary from a known location. Once the binary has been installed, the victim computer executes the malicious code and becomes a bot. In the connection phase, the bot attempts to establish a connection to the command and control server through a variety of methods, joining the botnet officially once this connection has been established. The maintenance phase is the last phase of the botnet lifecycle, bots are commanded to update their binaries, typically to defend against new attacks or to improve their functionality.

Leonard et al. divides a botnet's lifecycle into four phases: formation, C&C, attack and post-attack [5]. The attack phase is noted as a phase in the botnet lifecycle when the bot is actively performing malicious activities based on received instructions, while the post-attack phase is similar to the maintenance phase described in [4].

Many existing botnet detection techniques rely on detecting bot activity during the attack phase or during the initial infection / secondary injection phase. Typical detectors are based on traditional intrusion detection techniques, focusing on identifying botnets based on existing signatures of attacks by examining the behavior of underlying malicious activities.

In our work, we propose a method to detect the presence of peer to peer botnets not only during the attack phase, but also in the command and control phase. We examine the network behavior of a botnet at the level of the TCP/UDP flow, splitting it into multiple time windows and extracting from them a set of attributes which are then used to classify malicious (botnet) or non-malicious traffic using machine learning classification techniques. In particular, we compare the performance of the Bayesian Network classifier and a decision tree classifier using reduced error pruning (REPTree).

There are several advantages to detecting botnets based on their network flow characteristics. As a bot must communicate with the rest of the network during all phases after secondary injection, our approach may be used to detect the presence of a bot during a significant part of its life. Furthermore, detection based on network traffic characteristics is immune to encryption algorithms which may counter other approaches such as packet inspection and is computationally cheaper than those techniques. Additionally, by splitting individual flows into characteristic time windows, we may be able to detect bot activity quickly, before it has finished its tasks during the C&C or attack phases.

We organize the remainder of this paper in the following way: In Section 2, we provide an overview of existing botnet detection approaches and techniques. Section 3 provides our motivation and approach for the detection of botnets. In Section 4, we evaluate our approach using existing experimental datasets and compare the effectiveness of the two classification algorithms mentioned above which we have selected based on their performance and characteristics. Our concluding remarks and discussion of future work is provided in Section 5.

2 Related Work

A large collection of literature exists for the detection of botnets though interest towards the detection of peer to peer botnets has only recently emerged. Furthermore, botnet detection approaches using flow analysis techniques have only emerged in the last few years [6] and of these most examine flows in their entirety instead of smaller time intervals. Faily et al. classify botnet detection systems into four general types, signature-based detection, anomaly-based detection, DNS-based detection and mining-based detection [4]. Our focus will be on mining and anomaly based detection due to their increasing popularity.

Gu et al. proposed successively two botnet detection frameworks named BotHunter [7] and BotMiner [8].

BotHunter [7] is a system for botnet detection which correlates alarms from the Snort intrusion detection system with bot activities. Specifically, BotHunter exploits the fact that all bots share a common set of underlying actions as part of their lifecycle: scanning, infection, binary download, C&C and outbound scanning. BotHunter monitors a network and captures activity related to port scanning, outbound scanning and performs some payload analysis and malware activity detection based on Snort rules, and then uses a correlation engine to generate a score for the probability that a bot has infected the network. Like many behavior correlation techniques, BotHunter works best when a bot has gone through all phases of its

lifecycle, from initial exploit to outbound scan. BotHunter is also vulnerable to encrypted command and control channels that cannot be detected using payload analysis.

BotMiner [8] relies on the group behavior of individual bots within a botnet for its detection. It exploits the underlying uniformity of behavior of botnets and detects them by attempting to observe and cluster similar behavior being performed simultaneously on multiple machines on a network. BotMiner performs 'C-plane' clustering to first group network traffic behaviors which share similarities. Flows with known safe signatures (such as for some popular protocols) are filtered out of their list to improve performance. Once similar flows have been identified, BotMiner uses a second 'A-Plane' clustering technique which groups flows by the type of activities they represent using anomaly detection via Snort. By examining both the A-Plane and C-Plane, BotMiner correlates hosts which exhibit both similar network characteristics as well as malicious activity and in doing so identify the presence of a botnet as well as members of the network. Experimentally, BotMiner was able to achieve detection accuracies of 99% on several popular bot variants with a false positive rate around 1%.

Yu et al. proposed a data mining based approach for botnet detection based on the incremental discrete Fourier transform, achieving detection rates of 99% with a false positive rate of 14% [9]. In their work, the authors capture network flows and convert these flows into a feature stream consisting of attributes such as duration of flow, packets exchanged etc. The authors then group these feature streams using a clustering approach and use the discrete Fourier transform to improve performance by reducing the size of the problem via computing the Euclidean distance of the first few coefficients of the transform. By observing that individual bots within the same botnet tend to exhibit similar flow patterns, pairs of flows with high similarities and corresponding hosts may then be flagged as suspicious, and a traditional rule based detection technique may be used to test the validity of the suspicion.

Zeidanloo et al. proposed a botnet detection approach based on the monitoring of network traffic characteristics in a similar way to BotMiner. In their work, a three stages process of filtering, malicious activity detection and traffic monitoring is used to group bots by their group behavior. The approach divides the concept of flows into time periods of six hours and clusters these flow intervals with known malicious activity. The effects of different flow interval durations were not presented, and the accuracy of the approach is unknown.

All of the above mentioned group behavior clustering approaches requires that malicious activity be performed by the bots before detection may occur and therefore are unsuitable for early detection during the C&C phase of a bot's lifecycle. Additionally, similarity and group behavior detection strategies relies on the presence of multiple bots within the monitored network and are unreliable or non-functional if only a single infected machine is present on the network.

Livadas et al. proposed a flow based detection approach for the detection of the C&C traffic of IRC-based botnets, using several classifiers to group flow behavior [10]. Their approach generates a set of attributes from IRC botnet flows and classifies these flows using a variety of machine learning techniques. Using a Bayesian network classifier, they achieved a false negative rate between 10% to 20% and a false positive rate of 30% to 40% though their results may have been negatively affected by poor labeling criterion of data. They showed using their approach that there exists a

difference between botnet IRC chat behavior and normal IRC chat behavior and that it was possible to use a classifier to separate flows into these categories.

Wang et al. presented a detection approach of peer to peer based botnets (specifically the peer to peer Storm variants using the Kademlia based protocol) by observing the stability of control flows in initial time intervals of 10 minutes [11]. They developed an algorithm which measures the stability of flows and exploits the property that bots exhibit similar behavior in their command search and perform these tasks independently of each other and frequently. This differs from the usage of the protocol by a normal user which may fluctuate greatly with user behavior. They show that by varying parameters in their algorithm, they were able to classify 98% of Storm C&C data as 'stable', though a large percentage of non-malicious peer to peer traffic were also classified as such (with a false positive rate of 30%). Our own approach is similar to this research, though we seek to significantly increase our detection accuracy by introducing new attributes and by utilizing a machine learning algorithm.

3 Approach

3.1 Foundation

Early works in botnet detection are predominantly based on payload analysis methods which inspect the contents of TCP and UDP packets for malicious signatures. Payload inspection typically demonstrates very high identification accuracy when compared with other approaches but suffer from several limitations that are increasingly reducing its usefulness. Payload inspection techniques are typically resource intensive operations that require the parsing of large amounts of packet data and are generally slow. Additionally, new bots frequently utilize encryption and other methods to obfuscate their communication and defeat packet inspection techniques. Furthermore, the violation of privacy is also a concern in the payload analysis-based detection scheme.

A more recent technique, traffic analysis, seeks to alleviate some of the problems with payload inspection. Traffic analysis exploits the idea that bots within a botnet typically demonstrate uniformity of traffic behavior, present unique communications behavior, and that these behaviors may be characterized and classified using a set of attributes which distinguishes them from non-malicious traffic and techniques. Traffic analysis does not depend on the content of the packets and is therefore unaffected by encryption and there exists dedicated hardware which may extract this information with high performance without significantly impacting the network.

Typical traffic analysis based detection systems examine network traffic between two hosts in its entirety. While this approach is feasible for offline detection, it is not useful for the detection of botnet behavior in real time. A network flow between two hosts may run for a few seconds to several days, and in many instances it is desirable to discover botnet activity as soon as possible.

In this paper, we present a detection technique based on traffic analysis which allows us to identify botnet activity in real time by examining the characteristics of these flows in small time windows. We exploit some properties of botnet traffic in order to perform this detection with high confidence even when other non-malicious traffic is present on the network.

The uniformity of botnet communications and botnet behavior is well known and has been exploited by various architectures towards their detection [8] [9] [12] [13] [14]. Most of these techniques exploit this uniformity by monitoring the traffic behavior of a number of machines, and then identifying machines which are part of a botnet when they begin to simultaneously perform similar malicious actions. Other methods include observing the command and response characteristics of bots; in the BotSniffer architecture, Gu et al. detect individual bots by drawing a spatial-temporal correlation in the responses of bots to a specific command [13]. With this idea, we make the assumption that should there exist a unique signature for the flow behavior of a single bot, we can use this unique signature to detect many bots which are part of the same botnet.

Several studies have shown that it is possible to detect certain classes of network traffic simply by observing their traffic patterns. Jun et al. proposed a technique of detecting peer to peer traffic based on a set of network flow attributes [15]. While the research does not focus on computer security but instead traffic classification, they nevertheless show that it is possible to detect various classes of peer to peer applications (eMule, Kazaa, Gnutella) based on their unique flow attributes. We also observe that bots utilizing implementations of the Overnet / Kademlia p2p protocol as well as unique p2p implementations as those seen on the Waledac bot exhibit unique and specific message exchange characteristics, particularly when first joining their p2p networks [2] [16].

For our technique, we will analyze specifically the network flow characteristics of traffic on a network. For the purposes of our framework, we define a flow as a collection of packets being exchanged between two unique IP addresses using a pair of ports and utilizing one of several Layer 4 protocols. We observe the characteristics of a given flow by examining its traffic in a given time window T and make two observations about the size of the time window. First, if a time window is too small, we may fail to capture unique traffic characteristics that only become apparent over a longer period of time, and we may also introduce errors as the behavior of a flow may change over time. If a time window is too large, we cannot make a decision in our classification until the window has been met, which means that our time to detection will increase to an undesirably long period. Ultimately, the selection of a time window size will be based on a compromise between detection accuracy and speed.

In order to classify the flow characteristics, we compute a set of attributes for each time window which encodes relevant information about the behavior of the flow during that time window. The selection of our set of attributes is based on the observations we have made above, combined with our intuition of botnet messaging activities.

Operation of our detection framework consists of two phases. In the training phase, we provide our detectors with a set of known malicious and non-malicious data attribute vectors in order to train our classifiers in the identification of the two classes of data. Once complete, the system is placed in the detection phase, where it actively observes the network traffic and classifies the attribute vectors generated from active flows. When a set of attribute vectors has been classified as ‘malicious’ in the live data, the flows in question may be flagged as suspicious.

3.2 Attribute Selection

An attribute is some characteristic of a flow or a collection of flows in a given time window T which may be represented as a numeric or nominal value. Table 1 lists the set of 12 attributes we have selected for the purposes of our evaluation. Some attributes, such as the source and destination IP addresses and ports of a flow, may be extracted directly from the TCP / UDP headers, while others, such as the average length of packets exchanged in the time interval, require additional processing and computation. These attributes are then used as part of an attribute vector which captures the characteristics of a single flow for a given time interval.

Table 1. Selected network flow attributes

Attribute	Description
SrcIp	Flow source IP address
SrcPort	Flow source port address
DstIp	Flow destination IP address
DstPort	Flow destination port address
Protocol	Transport layer protocol or 'mixed'
APL	Average payload packet length for time interval.
PV	Variance of payload packet length for time interval.
PX	Number of packets exchanged for time interval.
PPS	Number of packets exchanged per second in time interval T
FPS	The size of the first packet in the flow.
TBP	The average time between packets in time interval.
NR	The number of reconnects for a flow
FPH	Number of flows from this address over the total number of flows generated per hour.

We selected our set of attributes based on the behavior of various well known protocols as well as the behavior of known botnets such as Storm, Nugache and Waledac. For example, we note that unlike normal peer to peer usage, bot communication may exhibit a more uniform behavior whereupon the bot queries for updates or instructions on the network continuously, resulting in many uniform sized, small packets which continuously occur. Another observation we may make is that for many protocols, the initial exchange of packets when a client joins a network tends to be unique and follows well defined behavior; this knowledge may allow us to assist in classification by capturing the characteristics of the initial packet exchange and carrying this information forward to subsequent time intervals for that flow. For instance, the first packet size attribute is obtained immediately when the initial flow has been established and is carried on to future time windows to assist in classification.

It should be noted that while included in our attribute list, the source and destination IP and port numbers for a flow may not be a very good attribute if the training data comes from a different network and uses different IP values. Typically we would like to use attributes which are universal to any network in order to provide for a more portable signature.

One final consideration for the selection of attributes is to provide some resistance to potential evasion techniques for bots. While no known bots today exhibit this evasion strategy, it is feasible that flow perturbation techniques could be used by a bot in an attempt to evade our analysis. A bot may, for example, inject random packets into its C&C communications in order to throw off correlations based on packet size. In order to mitigate some of these techniques, we measure the number of flows generated by a single address, and compare it with the number of total flows generated in some time period (in this case, an hour). This metric allows us to exploit the fact that most bots will generate more flows than typical non-malicious applications as it queries its C&C channels for tasks and carry out those tasks. We also measure the number of connections and reconnections a flow has made over time in case the bot attempts to randomly connect and disconnect to defeat our connection based metric. Like any service, it is desirable for a bot to be connected to its command and control channel as much as possible, and therefore any random disconnects a bot performs in order to defeat detection will naturally provide some mitigation against the bot's activities. Finally, it is possible to generate white lists of known IP addresses and services which help eliminate potential benign programs which may exhibit similar connection behavior to better isolate malicious applications. None of our proposed strategies are foolproof, but they serve to increase implementation complexity for the botmaster as well as provide natural detriments to the efficient operation of a botnet.

3.3 Classifier Selection

Many machine learning (ML) classification techniques exist which all attempt to cluster and classify data based on attribute sets. For purposes of our work, we would like to select classification techniques with a high performance in order to support real time detection goals while at the same time exhibiting high detection accuracy.

We have selected two popular classification techniques for our evaluation based on the above criteria, a Bayesian network classifier, and a decision tree classifier.

Bayesian networks (BN) are directed acyclic graphs where each node represents a domain variable, or in our case, a flow attribute, and each edge between nodes represents the probability of dependency. Given values assigned to other nodes, a BN may compute the conditional probability of one node occurring given values assigned to other nodes. These networks have been used in the classification of a variety of data, including network traffic data. Like most graphical classification techniques, Bayesian networks are easily visualized.

Decision tree based classifiers are a well known classification technique which exhibits desirably low computational complexity. In a decision tree, interior nodes represent input attributes with edges extending from them which correspond to

possible values of the attributes. These edges eventually lead to a leaf node which represents an output variable (in our case, whether a flow is malicious or non-malicious). Classification of an attribute vector simply consists of traversing the path from root to leaf by following edges which correspond to the appropriate values of the attributes in the attribute vector. Decision trees are learned via a partitioning process where the source attribute set is split into subsets based on a value test. This partitioning halts after a user-defined stopping criteria has been reached. For our evaluation, we select a decision tree using the Reduced Error Pruning algorithm (REPTree). This algorithm helps improve the detection accuracy of a decision tree with respect to noisy data, and reduces the size of the tree to decrease the complexity of classification.

4 Evaluation

4.1 Evaluation Dataset

There are considerable difficulties in obtaining real world datasets of botnet malicious activity. Many publicly available datasets consist of information collected from honeypots which may not reflect real-world usages. In a typical honeynet configuration, a honeypot is a machine dedicated for the collection of malicious data and typically is not used for other normal activities. In such a case, it is atypical to see non-malicious traffic within a honeypot network trace except in the smallest quantities, and such non-malicious data rarely reflect real world usage scenarios.

In order to evaluate our system, we attempt to generate a set of network traffic traces which contain both malicious and non-malicious traffic, including traffic from standard usage of popular networked applications. Malicious and non-malicious traffic are intermixed in a way that both types of traffic occur during the same time periods, and we label this data in order to evaluate the accuracy of our methods.

For this work, we obtained and used two separate datasets containing malicious traffic from the French chapter of the honeynet project involving the Storm and Waledac botnets, respectively [17]. Waledac is currently one of the most prevalent P2P botnets and is widely considered as the successor of the Storm botnet with a more decentralized communication protocol. Unlike Storm, which uses Overnet as a communication channel, Waledac utilizes HTTP communication and a fast-flux based DNS network exclusively.

To represent non-malicious, everyday usage traffic, we incorporated two different datasets, one from the Traffic Lab at Ericsson Research in Hungary [18] and the other from the Lawrence Berkeley National Laboratory (LBNL) [19]. The Ericsson Lab dataset contains a large number of general traffic from a variety of applications, including HTTP web browsing behavior, World of Warcraft gaming packets, and packets from popular bittorrent clients such as Azureus.

The LBNL is a research institute with a medium-sized enterprise network. The LBNL trace data consists of five datasets labeled D_0...D_4; Table 2 provides general information for each of the datasets.

Table 2. LBNL datasets general information

	D_0	D_1	D_2	D_3	D_4
Date	Oct 4, 04	Dec 15, 04	Dec 16, 04	Jan 6, 05	Jan 7, 05
Duration	10 min	1 hour	1 hour	1 hour	1 hour
Subnets	22	22	22	18	18
Hosts	2,531	2,102	2,088	1,561	1,558
Packets	18M	65M	28M	22M	28M

The recording of the LBNL network trace happened over three months period, from October 2004 to January 2005 covering 22 subnets. The dataset contains trace data for a variety of traffic which spans from web and email to backup and streaming media. This variety of traffic serves as a good example of day-to-day use of enterprise networks.

In order to produce an experimental dataset with both malicious and non-malicious traffic, we merged the two malicious datasets and the Erikson (non-malicious) dataset into a single individual trace file via a specific process depicted by Figure 2. First we mapped the IP addresses of the infected machines to two of the machines providing the background traffic. Second, we replayed all of the trace files using the TcpReplay tool on the same network interface card in order to homogenize the network behavior exhibited by all three datasets; this replayed data is then captured via wireshark for evaluation.

The final evaluation data produced by this process was further merged with all five LBNL datasets to provide one extra subnet to indeed simulate a real enterprise size

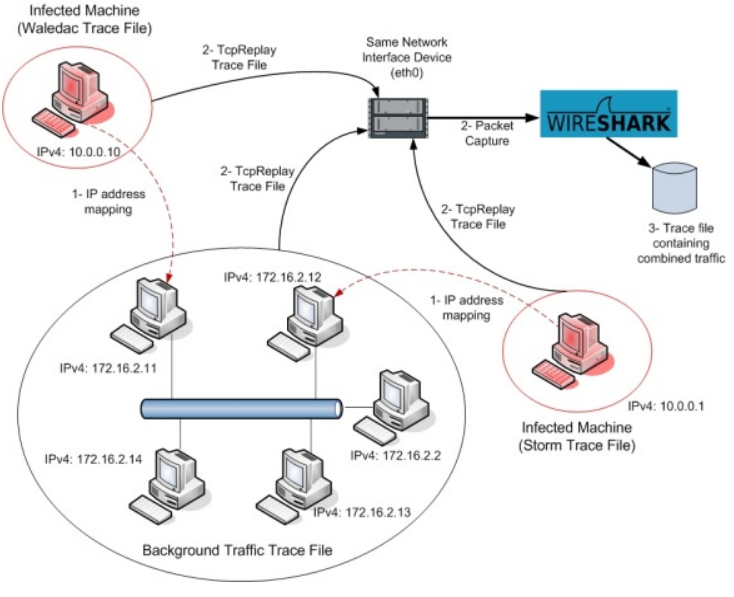

Fig. 1. Dataset merging process

network with thousands of hosts. The resulted evaluation dataset contains 22 subnets from the LBNL with non-malicious traffic and one additional subnet as illustrated in Figure 2 with both malicious and non-malicious traffic originating from the same machines.

4.2 Evaluation Results

We implemented our framework in Java and utilized the popular Weka machine learning framework and libraries for our classification algorithms [20]. Our program extracts from a given pcap file all flow information and then parses the flows into relevant attribute vectors for use in classification.

In all, there were a total of 1,672,575 network flows in our test set. The duration of the flows vary greatly, with some lasting less than a second and a few lasting more than a week. Of these flows, 97,043 (about 5.8%) were malicious, and the remainder non-malicious. From these flows, we generated 111,818 malicious attribute vectors, and 2,203,807 non-malicious attribute vectors. Each feature vector represents a 300 second time window in which at least 1 packet was exchanged. We consider malicious flow attribute vectors a vector which is extracted from a flow associated with the Storm or Waledec botnet data, and we considered all other attribute vectors as non-malicious, including peer to peer applications such as Bittorrent, Skype and e-Donkey.

To evaluate detection accuracy, we used the 10-fold cross-validation technique to partition our dataset into 10 random subsets, of which 1 is used for evaluation and 9 others are used for training. This process is repeated until all 10 subsets have been used as the testing set exactly once, while the remaining 9 folds are used for training. This technique helps us guard against Type III errors and gives us a better idea of how our algorithm may perform in practice outside of our evaluation data. The true and false positive rates of both the Bayesian Network and decision tree classifiers are listed in Table 2 and 3 respectively. The resulting detection values are an average of the results of the ten runs.

Table 3. Detection rate of Bayesian Network Classifier (attribute vectors identified)

Detection rate using Bayesian Network classifier (T = 300s)		
	True positive	False positive
Malicious	97.7%	1.4%
Non-Malicious	98.6%	2.3%

Table 4. Detection rate of REPTree classifier (attribute vectors identified)

Detection rate using decision tree classifier (REPTree) (T = 300s)		
	True positive	False positive
Malicious	98.3%	0.01%
Non-Malicious	99.9%	1.7%

As can be seen in the above tables, both the Bayesian Network classifier and the decision tree produced very high (above 90%) detection rates with a very low false positive rate. Between the two classifiers, the decision tree was more accurate, classifying 99% of all attribute vectors correctly while incorrectly identifying only 1% of all attribute vectors. These results indicate that there are indeed unique characteristics of the evaluation botnets when compared to everyday p2p and non-p2p traffic.

In terms of speed, the decision tree classifier was slightly slower than the Bayesian Network classifier, requiring 76.9 seconds for the full evaluation as compared to 56.1 seconds for the Bayesian Network.

In order to get some idea of the key discriminating attributes in our dataset, we use a correlation based attribute evaluator (CFS feature set evaluation) with best first search to generate a list of attributes with the highest discriminatory power while at the same time exhibiting low correlation with other attributes in the set. The algorithm generated a subset of four attributes, listed in Table 5, to be used as an optimized subset that may improve our performance without producing a large reduction in accuracy.

Table 5. Attribute subset from CFS subset evaluation

Feature	Description
PV	Variance of payload packet length for time interval.
PX	Number of packets exchanged for time interval.
FPS	The size of the first packet in the flow.
FPH	# flows per address / total flows

Tables 6 and 7 list the results of classification using only the above attribute subset. We can see that by reducing the number of attributes to three, the accuracy of the Bayesian network and REPTree classifiers both decreased slightly due to an increased false positive rate. Neither classifier with the reduced attribute set performed as well as the REPTree classifier with the full attribute set, though both very closely matched the best case's accuracy. In terms of performance, reducing the number of attributes allowed the Bayesian network classifier to classify all attribute vectors in 76% of the original time, while the REPTree classifier took 33% of the time. Table 8 shows the actual times for classifying all attribute vectors by the classifiers on both the full attribute set and the reduced set.

Table 6. Detection rate of Bayesian Network Classifier with reduced subset

Detection rate using Bayesian Network classifier (T = 300s)		
	True positive	False positive
Malicious	92.3%	1.9%
Non-Malicious	98.1%	7.7%

Table 7. Detection rate of REPTree classifier with reduced subset

Detection rate using decision tree classifier (REPTree) (T =300s)		
	True positive	False positive
Malicious	98.1%	2.1%
Non-Malicious	97.9%	1.9%

Table 8. Classifier performance (average training time)

Performance of classifiers	
Classifier	Time (seconds)
BayesNet	40.6
REPTree	29.4
BayesNet (subset)	11.22
REPTree (subset)	8.58

With the above results, we may conclude that both the Bayesian network classifier and the decision tree classifier are suitable for the building of a botnet detection framework based on flow attributes. We further observe that while maximum accuracy may be achieved with a sizable attribute vector, we may be able to detect unique characteristics in bot traffic based simply on their packet exchange behavior, in particular the variations in their flow behavior compared to standard network traffic and the first packet exchanged in any new flows created by such malicious traffic.

Finally, we examine the effects of varying the size of our time window on the accuracy of detection. Figures 3 and 4 show the effects of the time window size on the true and false positive rates for malicious flow detection. The best results were obtained when T = 180 seconds for both the REPTree and BayesNet classifiers, though very good results were obtained for all four time window sizes (10, 60, 180 and 300 seconds).

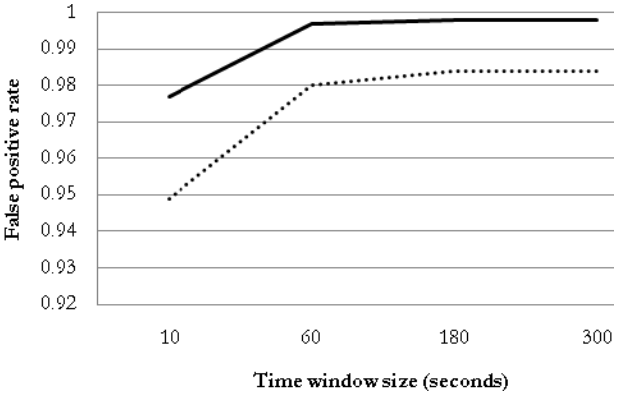

Fig. 2. Effects of time window size on true positive rate for REPTree *(solid line)* and BayesNet *(dotted line)* classifiers.

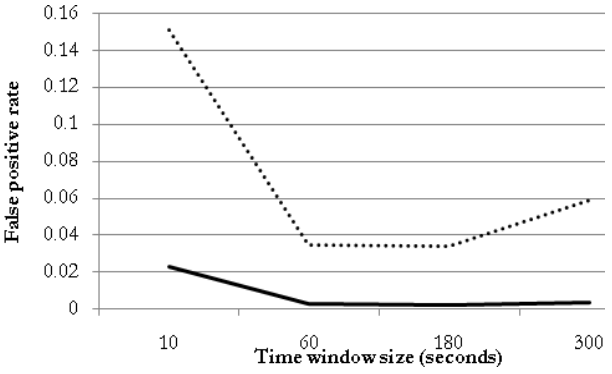

Fig. 3. Effects of time window size on false positive rate for REPTree *(solid line)* and BayesNet *(dotted line)* classifiers.

While both the performance and accuracy of our classifiers are satisfactory for real time detection, it must be noted that the system may be sensitive to new behaviors from bots implementing highly varied protocols. In order to guard against such a threat, a method for online training and continuous refinement of the classifiers must still be implemented.

4.3 Comparison with BotHunter

BotHunter [7] is one of the few botnet detection tools relevant to our work that is openly available. BotHunter mainly consist of a correlation engine that ties together alerts generated by Snort [21]. The tool consists of two custom plug-ins to snort called SLADE and SCADE. The SLADE plug-in mainly detects payload anomalies while the SCADE plug-in detects in/out bound scanning of the network. In addition to the two plug-ins, the tool includes a rule-set that is specifically designed to detect malicious traffic related to botnet activities such as Egg downloads and C&C traffic. The correlation engine ties all the alerts together and generates a report with the infection if any.

We used a recent version namely BotHunter version 1.6.0 and we tried to run it with our dataset. In practice, BotHunter is designed to run in live capture mode. However, it provides a script named runsnort.csh which allows running the tool offline against an existing dataset and generating an alert log file.

After running BotHunter against our dataset, the generated alerts indicated that there is a spambot in the dataset. More specifically, three alerts with “Priority 1” reporting the presence of botnet traffic were generated, however; the three alerts were all pointing to the same IP address. This IP address corresponds to a machine that was infected with Waledac botnet. BotHunter failed to detect the other machine that was infected with Storm botnet. In all, of the 97,043 unique malicious flows in the system, BotHunter was able to detect only a very small 56 flows. It is important to note that BotHunter itself does not work on the basis of flows, and the infected machine it detected is responsible for 17,006 malicious flows, (17% of the malicious traffic). It is important to also mention that BotHunter did not report any false positives.

While BotHunter does not expect or require that all phases of a bot lifecycle to be present in order to perform its detection, the fact that our dataset was missing the initial infection stages of the bot may have contributed to its poor detection performance.

5 Conclusion

In this paper we proposed a system for detecting bot activity in both the command and control and attack phases based on the observation of its network flow characteristics for specific time intervals. We emphasize the detection in the command and control phase because we would like to detect the presence of a bot before any malicious activities can be performed, and we use the concept of time intervals to limit the duration we would have to observe any particular flow before we may raise our suspicions about the nature of the traffic. We showed that using a decision tree classifier, we were able to successfully detect botnet activity with high accuracy by simply observing small portions of a full network flow, allowing us to detect and respond to botnet activity in real time. By comparing the true and false positive rates of our detector at various time window sizes, we have determined that a duration of 180 seconds provided the best accuracy of detection while a time window of 10 seconds was still able to produce an effective detector with a true positive rate of over 90% and a false positive rate under 5%.

There are limitations to our current approach that we hope to resolve in our future work. First, we recognize that our detection technique is based on the availability of existing malicious data and that in order for a detector to be truly robust we must develop a mechanism to evolve the classifiers to adapt to new threats. We are also aware that it is possible for a malicious botnet designer to obfuscate the network flow behavior of a bot in order to evade detection, even if such evasion would come at the expense of the effectiveness of a bot. To address these concerns, we are looking into the development of hybrid detectors which utilizes evolving classifiers along with our current approach, and defeat obfuscation by incorporating group behavior correlation.

References

1. Abu, R.M., et al.: A Multifaceted Approach to Understanding the Botnet Phenomenon. In: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement. ACM, New York (2006) ISBN:1-59593-561-4
2. Grizzard, J.B., et al.: Peer-to-Peer Botnets: Overview and Case Study. In: Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets. USENIX Association, Berkeley (2007)
3. Holz, T., et al.: 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. USENIX Association, Berkeley (2008)
4. Faily, M., Shahrestani, A., Ramadass, S.: A Survey of Botnet and Botnet Detection. In: Third International Conference on Emerging Security Information, Systems and Technologies (2009)

5. Leonard, J., Shouhuai, X., Sandhu, R.: A Framework for Understanding Botnets. In: International Workshop on Advances in Information Security. Fukuoka Institute of Technology, Fukuoka (2009)
6. Sperotto, A., et al.: An Overview of IP Flow-Based Intrusion Detection. IEEE Communications Surveys & Tutorial 12(3) (2010)
7. Gu, G., et al.: BotHunter: Deteting Malware Infection Through IDS-Driven Dialog Correlation. In: Proceedings of the 16th USENIX Security Symposium, pp. 167–182 (2007)
8. Gu, G., et al.: BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the 17th Conference on Security Symposium, San Jose (2008)
9. Yu, X., et al.: Online Botnet Detection Based on Incremental Discrete Fourier Transform. Journal of Networks 5(5) (2010)
10. Livadas, C., et al.: Using Machine Learning Techniques to Identify Botnet Traffic. In: 2nd IEEE LCN Workshop on Network Security, pp. 967–974 (2006)
11. Wang, B., et al.: Measuring Peer-to-Peer Botnets Using Control Flow Stability. In: International Conference on Availability, Reliability and Security, Fukuoka, p. 663 (2009), 978-1-4244-3572-2
12. Al-Duwairi, B., Al-Ebbini, L.: BotDigger: A Fuzzy Inference System for Botnet Detection. In: The Fifth International Conference on Internet and Applications and Services, Barcelona (2010)
13. Gu, G., Zhang, J., Lee, W.: BotSniffer: Detecting botnet command and control channels in network traffic. In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (2008)
14. Ricardo, V.-S., José, B.C.: Bayesian Bot Detection Based on DNS Traffic Similarity. In: Proceedings of the 2009 ACM Symposium on Applied Computing, pp. 2035–2041. ACM, Honolulu (2009), 978-1-60558-166-8
15. Jun, L., et al.: P2P Traffic Identification Technique. In: International Conference on Computational Intelligence and Security, Harbin, pp. 37–41 (2007), 0-7695-3072-9
16. Sinclair, G., Nunnery, C., Kang, B.B.: The Waledac Protocol: The How and Why. In: Proceeding of 4th IEEE International Conference on Malicious and Unwanted Software (2009)
17. The Honeynet Project, French Chapter | The Honeynet Project. The Honeynet Project, http://www.honeynet.org/chapters/france
18. Szabó, G., Orincsay, D., Malomsoky, S., Szabó, I.: On the Validation of Traffic Classification Algorithms. In: Claypool, M., Uhlig, S. (eds.) PAM 2008. LNCS, vol. 4979, pp. 72–81. Springer, Heidelberg (2008)
19. Lawrence Berkeley National Laboratory and ICSI., LBNL/ICSI Enterprise Tracing Project. LBNL Enterprise Trace Repository (2005), http://www.icir.org/enterprise-tracing
20. Witten, I.H., et al.: Weka: Practical Machine Learning Tools and Techniques (1999)
21. Roesch, M.: Snort - lightweight intrusion detection for networks. In: Proceedings of USENIX LISA 1999 (1999)