Flow-based Identification of Botnet Traffic by Mining Multiple Log Files

Mohammad M. Masud[1], Tahseen Al-khateeb[2], Latifur Khan[3]
Bhavani Thuraisingham[4], Kevin W. Hamlen[5]
Department of Computer Science, The University of Texas at Dallas
Richardson, TX 75080, USA
{[1]mehedy,[2]tahseen,[3]lkhan}@utdallas.edu
{[4]bhavani.thuraisingham,[5]hamlen}@utdallas.edu

***Abstract*—Botnet detection and disruption has been a major research topic in recent years. One effective technique for botnet detection is to identify Command and Control (C&C) traffic, which is sent from a C&C center to infected hosts (bots) to control the bots. If this traffic can be detected, both the C&C center and the bots it controls can be detected and the botnet can be disrupted. We propose a multiple log-file based temporal correlation technique for detecting C&C traffic. Our main assumption is that bots respond much faster than humans. By temporally correlating two host-based log files, we are able to detect this property and thereby detect bot activity in a host machine. In our experiments we apply this technique to log files produced by `tcpdump` and `exedump`, which record all incoming and outgoing network packets, and the start times of application executions at the host machine, respectively. We apply data mining to extract relevant features from these log files and detect C&C traffic. Our experimental results validate our assumption and show better overall performance when compared to other recently published techniques.**

***Keywords*- Malware, botnet, intrusion detection, data mining.**

I. Introduction

Botnets are emerging as "the biggest threat facing the internet today" [1] because of their enormous volume and sheer power. Botnets containing thousands of *bots* (compromised hosts) have been tracked by several different researchers [2], [3]. Bots in these botnets are controlled from a *Command and Control* (C&C) center, operated by a human *botmaster* or *botherder*. The botmaster can instruct these bots to recruit new bots, launch coordinated DDoS attack against specific hosts, steal sensitive information from infected machines, send mass spam emails, and so on. Fig. 1 illustrates a typical botnet architecture.

Numerous researchers are working hard to combat this threat and have proposed various solutions [4], [5], [2]. One major research direction attempts to detect the C&C center and disable it, preventing the botmaster from controlling the botnet. Locating the C&C center requires identifying the traffic exchanged between it and the bots. Our work adopts this approach by using a data mining based technique to identify temporal correlations between multiple log files. We maintain two different log files for each host machine: (i) a network packet trace or `tcpdump`, and (ii) an application execution trace or `exedump`. The `tcpdump` log file records all network packets that are sent/received by the host, and the `exedump` log file records the start times of application program executions on the host machine. Our main assumption is that bots respond to commands much faster than humans do. Thus, the command latency (i.e., the time between receiving a command and taking actions) should be much lower, and this should be reflected in the `tcpdump` and `exedump` log files.

Bot commands that have an observable effect upon the log files we consider can be grouped into three categories: those that solicit a response from the bot to the botmaster, those that cause the bot to launch an application on the infected host machine, and those that prompt the bot to communicate with some other host (e.g., a victim machine or a code server). This botnet command categorization strategy is explained in more detail in Section III. We apply data mining to learn temporal correlations between an incoming packet and (i) an outgoing packet, (ii) a new outgoing connection, or (iii) an application startup. Any incoming packet correlated with one of these logged events is considered a possible botnet command packet. Our approach is flow-based because rather than classifying a single packet as C&C or normal traffic, we classify an entire flow (or connection) to/from a host as C&C or normal. This makes the detection process more robust and effective. Our system is first trained with log files obtained from clean hosts and hosts infected with a known bot, then tested with logs collected from other hosts. The testing methodology is explained in detail in Section IV.

Our technique is different from other botnet detection techniques [5], [6], [2] in two ways. First, we do not impose any restriction on the communication protocol. Our approach should therefore also work with C&C protocols other than those that use IRC as long as the C&C traffic possesses the observable characteristics defined above. Second, we do not rely on command string matching. Thus, our method should work even if the C&C payloads are not available.

Our work makes two main contributions to botnet detection research. First, we introduce multiple log correlation for C&C traffic detection. We believe this idea could be successfully extended to additional application-level logs such as those that track process/service execution, memory/CPU utilization, and disk accesses. Second, we have proposed a way to classify botmaster commands into different categories, and we show

978-1-4244-2313-2/08/$25.00 ©2008 IEEE

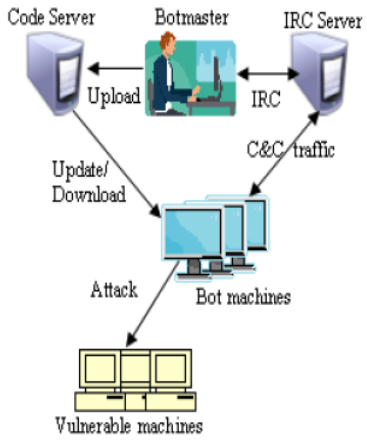

Fig. 1. A typical IRC-based botnet architecture

how to utilize these command characteristics to detect C&C traffic. An empirical comparison of our technique with another recent approach [5] shows that our strategy is more robust in detecting real C&C traffic.

The rest of the paper is organized as follows: Section II discusses related work on botnet detection. Section III discusses our system setup and data collection process. Section IV explains our botnet detection architecture and discusses the details of the detection process. Section V evaluates our system. Finally, Section VI concludes by summarizing our work and suggesting future research directions.

II. Related Work

Botnet defenses are being approached from at least three major perspectives: analysis, tracking, and detection. Barford and Yegneswaran [7] present a comprehensive analysis of several botnet codebases and discuss various possible defense strategies that include both reactive and proactive approaches. Grizzard et al. [4] analyze botnets that communicate using peer-to-peer networking protocols, concluding that existing defense techniques that assume a single, centralized C&C center are insufficient to counter these decentralized botnets.

Freiling et al. [3] summarize a general botnet-tracking methodology for manually identifying and dismantling malicious C&C centers. Rajab et al. [2] put this into practice for a specific IRC protocol. They first capture bot malware using a honeynet and related techniques. Captured malware is next executed in a controlled environment to identify the commands that the bot can receive and execute. Finally, *drone machines* are deployed that track botnet activity by mimicking the captured bots to monitor and communicate with the C&C server. Dagon et al. [8] track botnet activity as related to geographic region and time zone over a six month period. They conclude that botnet defenses such as those described above can be more strategically deployed if they take into account the diurnal cycle of typical botnet propagation patterns.

Our research presented in this article is a detection technique. Cooke et al. [9] discuss various botnet detection techniques and their relative merits. They conclude that monitoring C&C payloads directly does not typically suffice as a botnet-detection strategy because there are no simple characteristics of this content that reliably distinguish C&C traffic from normal traffic. However, Goebel and Holz [6] show that botnets that communicate using IRC can often be identified by their use of unusual IRC channels and IRC user nicknames. Livadas et al. [5] use additional features including packet size, flow duration, and bandwidth. Their technique is a two-stage process that first distinguishes IRC flows from non-IRC flows, and then distinguishes C&C traffic from normal IRC flows. While these are effective detection techniques for some botnets, they are specific to IRC-based C&C mechanisms and require access to payload content for accurate analysis and detection. In contrast, our method does not require access to botnet payloads and is not specific to any particular botnet communication infrastructure. Karasaridis et al. [10] consider botnet detection from an ISP or network administrator's perspective. They apply statistical properties of C&C traffic to mine large collections of network traffic for botnet activity. Our work focuses on detection from the perspective of individual host machines rather than ISP's.

III. Bot traffic analysis

In this section we describe our system setup, data collection process, and approach to categorizing bot commands.

A. System setup

We tested our approach on two different IRC-based bots—SDBot version 05a [11] and RBot version 0.5.1 [12]. The testing platform consisted of five virtual machines running atop a Windows XP host operating system. The host hardware consisted of an Intel Pentium-IV 3.2GHz dual core processor with 2GB RAM and 150GB Hard Disk. Each virtual machine ran Windows XP with 256 MB virtual RAM and 8GB virtual Hard Disk space. The five virtual machines played the role of a botmaster, a bot, an IRC server, a victim, and a code server, respectively. As with a typical IRC-based botnet, the IRC server served as the C&C center through which the botmaster issued commands to control the bot. The IRC server we used was the latest version of Unreal IRCd Daemon [13], and the botmaster's IRC chat client was MIRC. The code server ran Apache Tomcat, and contained different versions of bot malware code and other executables. The victim machine was a normal Windows XP machine. During the experiment the botmaster instructed the bot to target the victim machine with udp and ping attacks. All five machines were interconnected in an isolated network as illustrated in Fig. 1.

B. Data collection

Data collection was performed in three steps. First, we implemented a client for the botmaster that automatically sent all possible commands to the bot. Second, we ran Windump [14] to generate a `tcpdump` log file, and ran our own implementation of a process tracer to generate a `exedump` log file. Third, we ran each bot separately on a fresh virtual machine, collected the resulting traces from the log files, and

TABLE I
SDBOT AND RBOT COMMAND CHARACTERISTICS

Observable effects	Commands					
	addalias	about	execute	udp	cmd	download
Bot-app	×	×	✓	×	✓	✓
Bot-response	×	✓	×	✓	✓	✓
Bot-other	×	×	×	✓	×	✓

then deleted the infected virtual machine. Traces were also collected from some uninfected machines connected to the internet. Each trace spanned a 12-hour period. The `tcpdump` traces amounted to about 3GB in total. Finally, these traces were used for training and testing.

C. Bot command categorization

Not all bot commands have an observable effect upon the log files we consider. We say that a command is *observable* if it matches one or more of the following criteria:

1) **Bot-response:** The command solicits a reply message from the bot to the C&C center. This reply is logged in the `tcpdump`. For example, the SDbot commands `about` and `sysinfo` are observable according to this criterion.
2) **Bot-app:** The command causes the bot to launch an executable application on the infected host machine. The application start event will be logged in the `exedump`. The `execute` command from SDbot is an example of such a command.
3) **Bot-other:** The command causes the bot to contact some host other than the C&C center. For example, the command might instruct the bot to send UDP packets as part of a DoS attack, send spam emails to other hosts, or download new versions of bot malware from a code server. Such events are logged in the `tcpdump`.

Some of the SDBot and RBot commands are listed in Table 1 and categorized using the above mentioned criteria. For a comprehensive description of these commands, please refer to [11], [12].

IV. ARCHITECTURE

Data collected via the procedure described in Section III was used for training and testing using the architecture illustrated in Fig. 2. For training, we first label each flow—i.e., each $(ip{:}port, ip'{:}port')$ pair—as a bot flow (conversation between a bot and its C&C center), or a normal flow (all other connections). Second, we compute several packet-level features (detailed below) for each incoming packet. Third, we compute flow-level features for each flow by aggregating the packet-level features. Finally, these flow-level features are used to train a classifier and obtain a classification model. For testing, we take an unlabeled flow and compute its flow-level features in the same way. Then we test the feature values against the classification model and label it a normal flow or a bot flow. The feature extraction process is explained next.

A. Feature Extraction

First we discuss the packet-level features, and then discuss the flow-level features. The intuitive idea behind these features is that human response to a command/request (e.g., a request to send a file or execute an application by his peer) should be much slower than a bot. In what follows, we refer to a packet as *incoming* if its destination is the host being monitored, and as *outgoing* if it originates from the monitored host.

a) Packet-level features.: The packet-level features we consider can be summarized as follows:

- **Bot-response (BR)** (boolean-valued): An incoming packet possesses this feature if it originated from some $ip{:}port$ and there is an outgoing packet to the same $ip{:}port$ within 100 ms of arrival of the incoming packet. This indicates that it is a potential command packet. The 100 ms threshold has been determined by our observation of the bots. We will refer to these incoming packets as *BR packets*.
- **BRtime** (real-valued): This feature records the time difference between a BR packet and its corresponding outgoing packet. This is an important characteristic of a bot.
- **BRsize** (real-valued): This feature records the length (in KB) of a BR packet. We observe that command packets typically have lengths of 1KB or less, whereas normal packets have unbounded size.
- **Bot-other (BO)** (boolean-valued): An incoming packet possesses this feature if it originated from some $ip{:}port$ and there is an outgoing packet to some $ip'{:}port'$ within 200 ms of the arrival of the incoming packet, where $ip' \neq ip$. This is also a potential command packet. The 200 ms threshold has also been determined by our observation of the bots. We will refer to these incoming packets as *BO* packets.
- **BODestMatch** (boolean-valued): A BO packet possesses this feature if outgoing destination ip' is found in its payload. This indicates that the BO packet is possibly a command packet that tells the bot to establish connection with host ip'.
- **BOtime** (real-valued): This feature records the time difference between a BO packet and its corresponding outgoing packet. This is also an important characteristic of a bot.
- **Bot-App (BA)** (boolean-valued): An incoming packet possesses this feature if an application starts on the host machine within 3 seconds of arrival of the incoming packet. This indicates that it is potentially command packet that instructs the bot to run an application. The 3

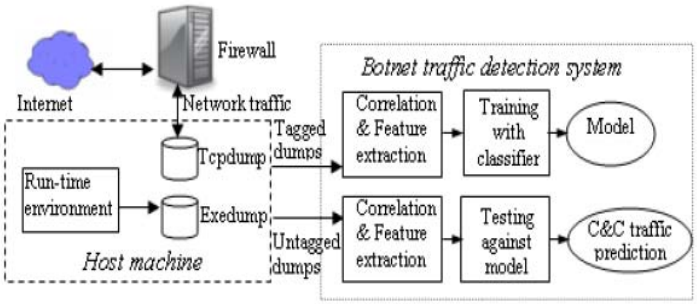

Fig. 2. System architecture

TABLE II
FLOW-LEVEL FEATURE SET

Feature	Description
AvgPktLen **VarPktLen**	Average and Variance of length of packets in KB
Bot-app	Number of BA packets as percentage of total packets
AvgBAtime **VarBAtime**	Average and Variance of BAtime of all BA packets
Bot-reply	Number of BR packets as percentage of total packets
AvgBRtime **VarBRtime**	Average and Variance of BRtime of all BR packets
AvgBRsize **VarBRsize**	Average and Variance of BRsize of all BR packets
Bot-other	Number of BO packets as percentage of total packets
AvgBOtime **VarBOtime**	Average and Variance of BOtime of all BO packets

second threshold has been determined by our observation of the bots. We will refer to these incoming packets as *BA* packets.

- **BAtime** (real-valued): This feature records the time difference between receiving a BA packet and the launching of the corresponding application.
- **BAmatch** (boolan-valued): A BA packet possesses this feature if its payload contains the name of the application that was launched.

b) Flow-level features.: As explained earlier, the flow-level features of a flow are the aggregations of packet-level features in that flow. They are summarized in Table 2. All flow-level features are real-valued. Also note that we do not use any flow-level feature that requires payload analysis.

B. Log file Correlation

Fig. 3 shows an example of multiple log file correlation. Portions of the `tcpdump` (left) and `exedump` (right) log files are shown in this example, side by side. Each record in the `tcpdump` file contains the packet number (No), arrival/departure time (Time), source and destination addresses (Src/Dest), and payload or other information (Payload/Info). Each record in the `exedump` file contains two fields: the process start time (Start Time), and process name (Process). The first packet (#10) shown in the `tcpdump` is a command packet that instructs the bot to download an executable from the code server and run it. The second packet (#11) is a response from the bot to the botmaster, so the command packet is a BR packet having $BRtime = 1ms$. The bot quickly establishes a TCP connection with the code server (other host) in packets #12–14. Thus, the command packet is also a BO packet having $BOtime = 7ms$ (the time difference between the incoming command and the first outgoing packet to another host). After downloading, the bot runs the executable `mycalc.exe`. Thus, this command packet is also a BA packet having $BAtime = 2.283s$.

C. Classification

We use a Support Vector Machine (SVM), Bayes Net, decision tree (J48), Naïve Bayes, and Boosted decision tree (Boosted J48) for the classification task. In our previous work [15] we have found that each of these classifiers demonstrates good performance for malware detection problems. Specifically, SVM is robust to noise and high dimensionality and can be fine-tuned to perform efficiently on a specific domain. Decision trees have a very good feature-selection capability and are much faster than many other classifiers both in training and testing time. Bayes Nets are capable of finding the interdependencies between different attributes. Naïve Bayes is also fast, and performs well when the features are independent of one another. Boosting is particularly useful because of its ensemble methods. Thus, each of these classifiers has its own virtue. In a real deployment, we would actually use the best among them.

D. Packet Filtering

One major implementation issue related to examining the packet traces is the large volume of traffic that needs to be scanned. We try to reduce unnecessary scanning of packets by filtering out the packets that are not interesting to us, such as the TCP handshaking packets (`SYN`,`ACK`,`SYNACK`) and NetBios session request/response packets.

V. EVALUATION

For evaluation, we tag all the flows as either bot flows or normal flows depending on whether the flow is between a bot and its C&C center or not. Then we extract feature-values

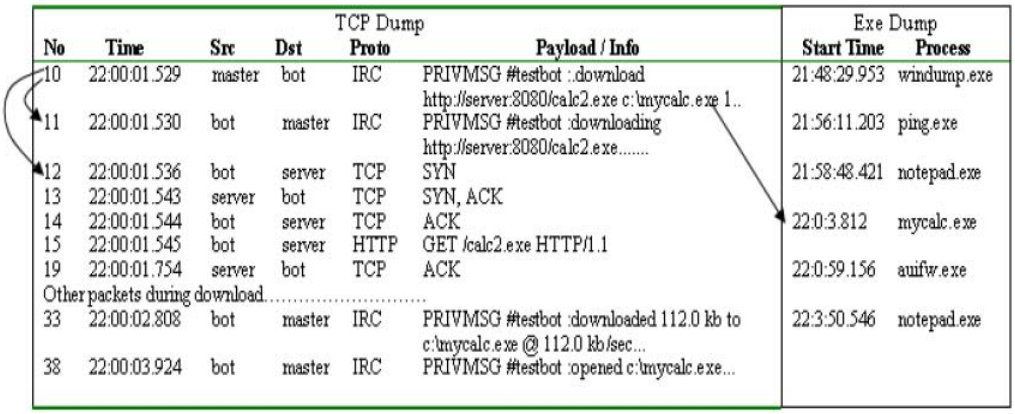

Fig. 3. Multiple log file correlation

TABLE III
PERFORMANCES OF DIFFERENT CLASSIFIERS ON FLOW-LEVEL FEATURES

Dataset	Metric	Boosted-J48	Bayes-Net	Naive-Bayes	J48	SVM
SDBot	ACC%	98.9	99.0	98.9	98.8	97.8
	FP%	1.5	1.3	1.5	1.6	3.0
	FN%	0.0	0.0	0.0	0.0	0.0
RBot	ACC%	98.8	96.4	95.2	96.4	96.4
	FP%	1.5	3.0	3.1	3.2	3.0
	FN%	0.0	4.2	6.5	4.0	4.2

TABLE IV
COMPARING PERFORMANCES BETWEEN OUR METHOD (TEMPORAL) AND THE METHOD OF LIVADAS ET AL. ON THE COMBINED DATASET

Method	Metric	Boosted-J48	Bayes-Net	Naive-Bayes	J48	SVM
Temporal	ACC%	99.9	99.5	99.1	99.2	99.1
Livadas	ACC%	97.0	99.7	97.1	97.5	99.0
Temporal	FP%	0.0	0.0	0.0	0.0	0.0
Livadas	FP%	0.3	0.0	0.0	0.0	0.0
Temporal	FN%	0.2	0.9	1.9	1.7	1.9
Livadas	FN%	6.5	0.6	6.3	5.9	2.1

for each flow using the technique described in Section IV-A. Finally, we apply five-fold cross validation on the data and report the accuracy and false alarm rates. We use the Weka ML toolbox [16] for classification.

A. Performance on different data sets

Table 3 reports the classification accuracies (ACC), false positive rates (FP), and false negative rates (FN) for each of the classifiers for different datasets. The datasets SDBot and RBot correspond to those where the bot-flows are generated only from SDBot and RBot, respectively; and normal flows are generated from uninfected machines. The results are obtained by applying five-fold cross validation on the datasets. Boosted J48 has the best detection accuracy (98.8%) for RBot, whereas Bayes Net has the best detection accuracy (99.0%) for SDBot. However, it is evident that Boosted J48 is less dataset-sensitive since it performs consistently on both datasets, and Bayes Net is only 0.1% better than Boosted J48 for the SDBot dataset. Thus, we conclude that BoostedJ48 has overall better performance than other classifiers. This is also supported by the results presented next.

B. Comparison with other techniques

We also compare our technique with another machine-learning technique applied by Livadas et al. [5]. They extract several flow-based features, such as a histogram of packet sizes, flow duration, bandwidth etc., but these are different from our feature set. They first identify IRC flows and then detect bot flows in the IRC flows. We don't need to identify IRC flows to detect C&C traffic using our analysis, but in order to perform a fair comparison we filter out non-IRC flows. We then extract their optimal set of features from the filtered data, apply five-fold cross validation, and report the accuracy and false alarm rates.

The rows labeled "Temporal" and "Livadas" in Table 4 report the classification accuracies (ACC), false positive rates (FP), and false negative rates (FN) of our technique and the technique of Livadas et al. [5], respectively. The comparison reported is for the combined dataset that consists of bot-flows from both SDBot and RBot infected machines, and all the normal flows from uninfected machines (with non-IRC flows filtered out). We see that Temporal performs consistently across all classifiers having accuracy $> 99\%$, whereas Livadas has $\leq 97.5\%$ accuracy in three classifiers and shows slightly better accuracy (0.2% higher) than Temporal only with Bayes Net. Bayes Net tends to perform well on a feature set if there are dependencies among the features. Since it is likely that there are dependencies among the features used by Livadas, we infer that the overall detection accuracy of Livadas is probably sensitive to classifiers, whereas Temporal is robust to all classifiers. Additionally, Temporal outperforms Livadas in false negative rates for all classifiers except Bayes Net. Finally, we again find that BoostedJ48 has the best performance among all classifiers, so we conclude that our Temporal method with BoostedJ48 has the best overall performance.

Fig. 4 presents the *receiver operating characteristic* (ROC) curves corresponding to the combined dataset results. ROC curves plot true positive rate against false positive rate. An

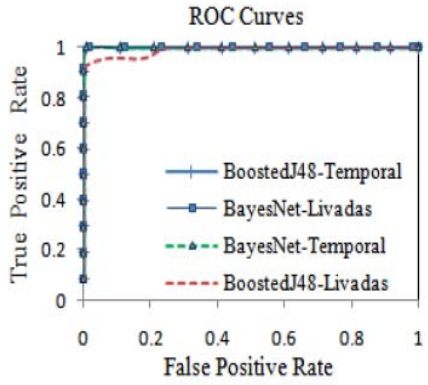

Fig. 4. ROC curves of Bayes Net and Boosted J48 on the combined data

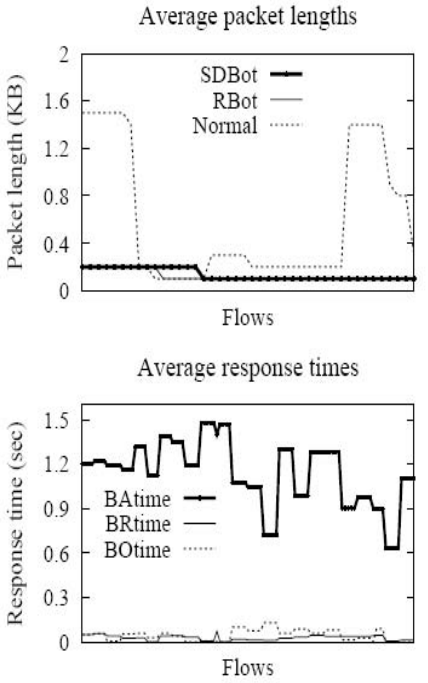

Fig. 5. Flow summary statistics. (above): Average packet lengths of normal and bot-flows, (below): Average *BRtime*, *BOtime*, and *BAtime* of bot-flows

ROC curve is better if the *area under the curve* (AUC) is higher, which indicates a higher probability that an instance will be correctly classified. In this figure, the ROC curve labeled as "Bayes Net-Livadas" corresponds to the ROC curve of Bayes Net on the combined data set for the Livadas et al. technique, and so on. We see that all of the ROC curves are almost co-incidental, except BoostedJ48-Livadas, which is slightly worse than the others. The AUC of "BoostedJ48-Livadas" is 0.993, whereas the AUC of all other curves are greater than or equal to 0.999.

C. Further analysis

Fig. 5 shows statistics of several features. The upper chart plots the average packet length (in KB) of each flow that appears in the dataset. Bot-flows and normal flows are shown as separate series. A data point (X, Y) represents the average packet length Y of all packets in flow X of a particular series (bot-flow or normal). It is clear from the chart that bot-flows have a certain packet length ($\leq 0.2KB$), whereas normal flows have rather random packet lengths. Thus, our assumption about packet lengths is validated by this chart. The lower chart plots three different response times: Bot-response time (*BRtime*), Bot-other time (*BOtime*), and Bot-app time (*BAtime*) for each bot-flow. It is evident that average *BRtime* is less than 0.1 second, average *BOtime* is less than 0.2 seconds and average *BAtime* is between 0.6 and 1.6 seconds. The threshold values for these response times were chosen according to these observations.

VI. Conclusion

We presented the novel idea of correlating multiple log files and applying data mining for detecting botnet C&C traffic. Our idea is to utilize the temporal correlation between two different log files: `tcpdump`, and `exedump`. The `tcpdump` file logs all network packets that are sent/received by a host, whereas the `exedump` file logs the start times of application program executions on the host. We implement a prototype system and evaluate its performance using five different classifiers: support vector machines, decision trees, Bayes Nets, Boosted decision trees, and Naïve Bayes. Comparison with another technique by Livadas et al. [5] for C&C traffic detection shows that our method has overall better performance when used with a Boosted decision tree classifier. The technique used by Livadas et al. first identifies IRC flows and then detects botnet traffic from the IRC flows. Our technique is more general because it does not need to identify IRC traffic and is therefore applicable to non-IRC botnet protocols, as long as certain realistic assumptions about the command-response timing relationships (detailed in Section III) remain valid.

In future work we intend to apply this temporal correlation technique to more system level logs such as those that track process/service executions, memory/CPU utilization, disk reads/writes, and so on. We also would like to implement a real-time C&C traffic detection system using our approach.

References

[1] T. Ferguson, "Botnets threaten the internet as we know it," *ZDNet Australia*, April 2008.

[2] M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proc. 6th ACM SIGCOMM Conference on Internet Measurement (IMC'06)*, 2006, pp. 41–52.

[3] F. Freiling, T. Holz, and G. Wicherski, "Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks," in *Proc. 10th European Symposium On Research In Computer Security (ESORICS)*, vol. Lecture Notes in Computer Science 3676, September 2005, pp. 319–335.

[4] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *Proc. 1st Workshop on Hot Topics in Understanding Botnets*, 2007, p. 1.

[5] C. Livadas, B. Walsh, D. Lapsley, and W. Strayer, "Using machine learning techniques to identify botnet traffic," in *Proc. 31st IEEE Conference on Local Computer Networks (LCN'06)*, November 2006, pp. 967–974.

[6] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by irc nickname evaluation," in *Proc. 1st Workshop on Hot Topics in Understanding Botnets*, 2007, p. 8.

[7] P. Barford and V. Yegneswaran, *An Inside Look at Botnets*, ser. Advances in Information Security. Springer, 2006.

[8] D. Dagon, C. Zou, and W. Lee, "Modeling botnet propagation using time zones," in *Proc. 13th Network and Distributed System Security Symposium (NDSS'06)*, 2006.

[9] E. Cooke, F. Jahanian, and D. McPherson, "The zombie roundup: Understanding, detecting, and disrupting botnets," in *Proc. Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI'05)*, 2005, p. 6.

[10] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *Proc. 1st Workshop on Hot Topics in Understanding Botnets*, 2007, p. 7.

[11] (2006) SDBOT information webpage. [Online]. Available: www.megasecurity.org/trojans/s/sdbot/Sdbot0.5a.html.

[12] (2006) RBOT information webpage. [Online]. Available: http://jarryd.onestop.net/rxbot-howto.html.

[13] (2007) The Unreal IRC Daemon website. [Online]. Available: http://www.unrealircd.com/.

[14] (2007) The Windump website. [Online]. Available: http://www.winpcap.org/windump/.

[15] M. M. Masud, L. Khan, and B. Thuraisingham, "A scalable multi-level feature extraction technique to detect malicious executables," *Information Systems Frontiers*, vol. 10, no. 1, pp. 33–45, March 2008.

[16] (2008) The WEKA Data Mining with Open Source Software website. [Online]. Available: http://www.cs.waikato.ac.nz/ml/weka/.